



IMPROVED IMAGE PROCESSING ARCHITECTURE

by Gilbert P. Hyatt

08/464034

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application  
is a continuing application continuing from  
parent application IMPROVED IMAGE PROCESSING ARCHITECTURE S/N

07/289,355 filed on December 22, 1988 now pending;

where this parent application S/N 07/289,355 is a continuing application continuing from parent application IMPROVED IMAGE PROCESSING ARCHITECTURE S/N 06/663,094 filed on October 19, 1984

where this parent application S/N 06/663,094 is a continuation in part of two patent applications:

1. IMPROVED MEMORY ARCHITECTURE HAVING MULTI-DIMENSIONAL ADDRESSING by Gilbert P. Hyatt Serial No. 06/661,649 filed on October 17, 1984 and

2. IMPROVED MEMORY ARCHITECTURE HAVING A MULTIPLE BUFFER OUTPUT ARRANGEMENT by Gilbert P. Hyatt Serial No. 06/662,211 filed on October 18, 1984;

where the benefit of the filing dates of all of the above ancestor applications is claimed in accordance with 35 USC 120 and other authorities therefor and where ancestor applications Serial No. 06/661,649; Serial No. 06/662,211; and Serial No. 06/663,094 are herein incorporated by reference.

## BACKGROUND OF THE INVENTION

### FIELD OF THE INVENTION

The field of the present invention is display and machine vision systems and, in particular, image processing systems.

### PRIOR ART

The prior art in display systems includes graphic systems, image processing systems, and many other types. These systems are typically large, expensive, and do not operate in real time. Compact avionic packaging does provide small systems. Real time display systems are available in the graphics market, such as the CT-5 from Evans and Southerland, and in the image processing market in the form of broadcast systems, such as the Mirage from MCI Quantel, and in the form of moving map display systems, such as from Bendix and Harris.

### SUMMARY OF THE INVENTION

The present invention is generally directed to improved processing systems and, in particular, provides improvements in memory, processor, software, and control architectures and in software and hardware designs. A system architecture is provided that provides flexibility, performance, and efficiency. A geometric processor is provided that implements rotation, translation, expansion, compression, warping, 3D-perspective, vector generation, and other features in a high performance and efficient manner; such as by using a window implementation and by providing the flexibility of software control. A spatial processor is provided that provides smoothing, anti-aliasing, and other features. Memory architectures and designs are provided that increase performance and efficiency, including a memory map and a buffer memory. Programs are provided that enhance flexibility and capability, yet preserve the high performance of the hardware. For example, efficient geometric initial condition generation and time domain interpolation enhance performance. Many other novel and valuable features are disclosed.

#### BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention may be obtained from a consideration of the detailed description hereinafter taken in conjunction with the drawings, which are briefly described below.

Fig 1, comprising Figs 1A to 1P, provides block diagrams of various configurations of the present invention together with various flow diagrams and image diagrams; where Fig 1A is a block diagram representation of one configuration of the system of the present invention; Fig 1B is a block diagram of a geometric module configuration; Fig 1C is a block diagram of a single channel configuration of one configuration of the present invention in accordance with a reduced implementation of Fig 1A; Fig 1D is a block diagram of a geometric processor module in accordance with Fig 1A; Fig 1E is a block diagram of a spatial processor module in accordance with Fig 1A; Fig 1F is a block diagram of input sources, input interfaces, and input multiplexers/demultiplexers in accordance with Fig 1A; Fig 1G is a block diagram of an output interface, output multiplexers/demultiplexers, and output devices in accordance with Fig 1A; Fig 1H is a block diagram of an alternate configuration of the system of the present invention as implemented in an experimental system; Fig 1I is a block diagram of another alternate configuration of the system of the present invention; Fig 1J is a block diagram of still another configuration of the system of the present invention; Fig 1K is an image processing window arrangement; Fig 1L is a block diagram of still another configuration of the system of the present

invention; Fig 1M is an image diagram showing geometric processing; Fig 1N is an image diagram showing image translation; Fig 1O is a flow diagram showing operation of one geometric processor configuration; and Fig 1P is another block diagram representation of an arrangement for implementing the system of the present invention.

Fig 2, comprising Figs 2A to 2M, describes an image processor configuration with window diagrams, flow diagrams, and logic diagrams; where Fig 2A is a diagram of the memory hierarchy associated with window processing; Fig 2B is a diagram of window processing in accordance with Fig 2A; Fig 2C is one configuration of a window diagram showing rotation about the window center; Fig 2D is a configuration of a window diagram showing rotation about an offset from the window center; Fig 2E is another configuration of a window diagram showing rotation about the window center; Fig 2F is a configuration of a window diagram showing rotation about an offset from the window center; Fig 2G is still another configuration of a window diagram showing rotation about the window center; Fig 2H is a flow diagram showing geometric processing in accordance with the window arrangement of Fig 2G; Fig 2I is a schematic and block diagram representation of address generators for providing geometric processing; Fig 2J is a ^  
diagrammatic representation of expansion processing; Fig 2K is a ^  
diagrammatic representation of compression processing; Fig 2L is a ^  
diagrammatic representation of address generator scaling; Fig 2M is a block diagram representation of an arrangement for implementing the addressing and architecture of the memory of the

present invention.

Fig 3, comprising Figs 3A to 3C, showing image memory operations.

Fig 4, comprising Figs 4A to 4B, is a diagrammatic representation of address generators; where Fig 4A is a diagram of an address generator partitioned into an X-address component and a Y-address component and Fig 4B is a diagram of an address generator concatenating an X-address component and a Y-address component.

Fig 5, comprises Figs 5A to 5D: where Fig 5A is a block diagram representation of a spatial filter arrangement; Fig 5B is a block diagram representation of a sum-of-the-products arrangement that can be used with the arrangement of Fig 5A; Fig 5C is a block diagram of a 3-channel sum-of-the-products arrangement; and Fig 5D is a block diagram of a multiple channel buffer memory.

Fig 6, comprises Figs 6A to 6AH: where Fig 6A is a block diagram representation of a system configuration for implementation of the present invention; Fig 6B is a detailed schematic diagram of clock steering logic; Fig 6C is a detailed schematic diagram of clock gating logic; Fig 6D is a detailed schematic diagram of control logic; Fig 6E is a block diagram of a configuration for implementing the memory of the present invention; Fig 6F is a detailed schematic representation of logic for addressing and scanning-out memory information in accordance with the memory of Fig 6E; Figs 6G to 6J are detailed block diagram representations in accordance with the memory of Fig 6E; Figs 6K to 6N are detailed schematic diagram representations in

accordance with the memory of Figs 6F and Figs 6G to 6J; Figs 6O and 6P are detailed schematic diagram representations of one configuration of an address generator that can be used in the system of the present invention; Figs 6Q and 6R are detailed schematic diagram representations of another configuration of an address generator that can be used in the system of the present invention; Fig 6S is a detailed schematic diagram representation of a video DAC channel; Fig 6T is a detailed schematic diagram representation of a video synchronization pulse generator and clock pulse generator; Fig 6U is a detailed schematic diagram representation of joystick interface logic; Fig 6V is a detailed schematic diagram representation of joystick analog to digital converters; Fig 6W is a detailed schematic diagram representation of an address counter arrangement for loading a multiple buffer memory and related logic; Fig 6X is a detailed schematic diagram representation of an address counter arrangement for unloading a multiple buffer memory and related logic; Fig 6Y is a detailed schematic diagram representation of a first channel address multiplexer and RAM arrangement for a multiple channel buffer memory; Fig 6Z is a detailed schematic diagram representation of a first channel RAM input/output multiplexer arrangement for a multiple channel buffer memory; Fig 6AA is a detailed schematic diagram representation of a second channel address multiplexer and RAM arrangement for a multiple channel buffer memory; Fig 6AB is a detailed schematic diagram representation of a second channel RAM input/output multiplexer arrangement for a multiple channel buffer memory; Fig 6AC is a detailed schematic diagram

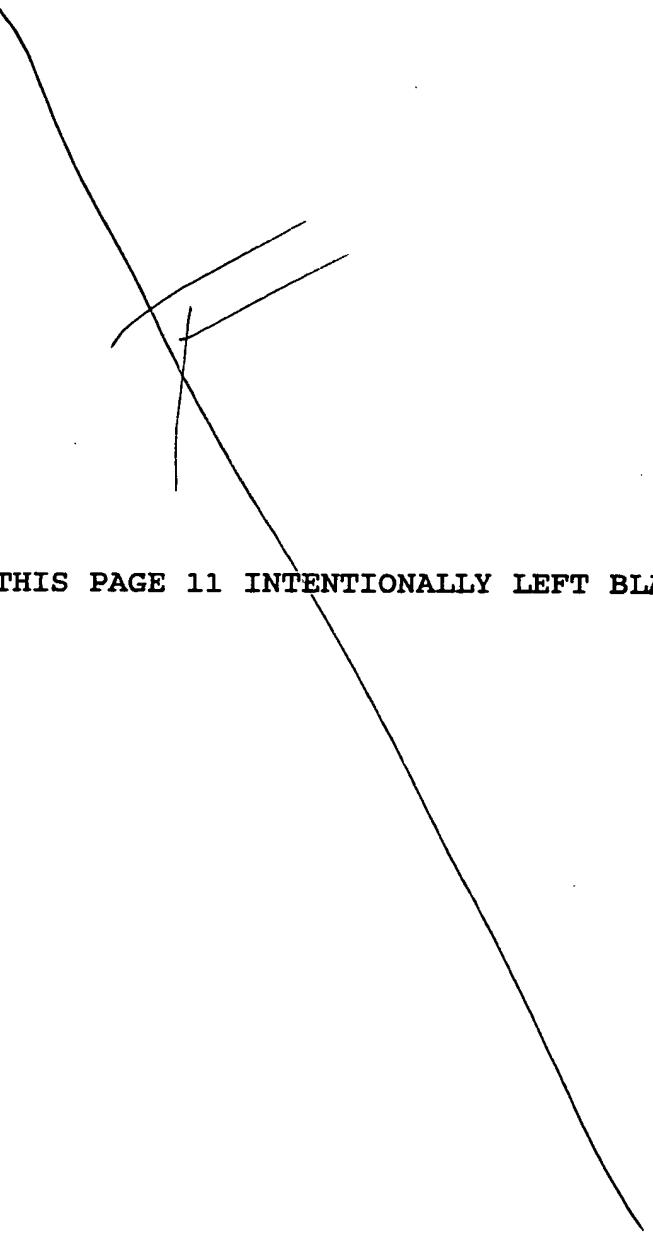
representation of a third channel address multiplexer and RAM arrangement for a multiple channel buffer memory; Fig 6AD is a detailed schematic diagram representation of a third channel RAM input/output multiplexer arrangement for a multiple channel buffer memory; Fig 6AE is a detailed schematic diagram representation of a forth channel address multiplexer and RAM arrangement for a multiple channel buffer memory; Fig 6AF is a detailed schematic diagram representation of a forth channel RAM input/output multiplexer arrangement for a multiple channel buffer memory; Fig 6AG is a detailed schematic diagram representation of a 9-pixel kernel register arrangement for spatial processing; and Fig 6AH is a detailed schematic diagram representation of a weight RAM arrangement for spatial processing.

Fig 7, comprising Figs 7A to 7D, show database memory and filtering configurations; where Figs 7A to 7C show mosaic arrangements and Fig 7D is a detailed flow diagram representation of the BASIC PROGRAM LISTING FTR.ASC for filtering images.

Fig 8 is a block diagram representation showing one configuration of a system for processing large database images.

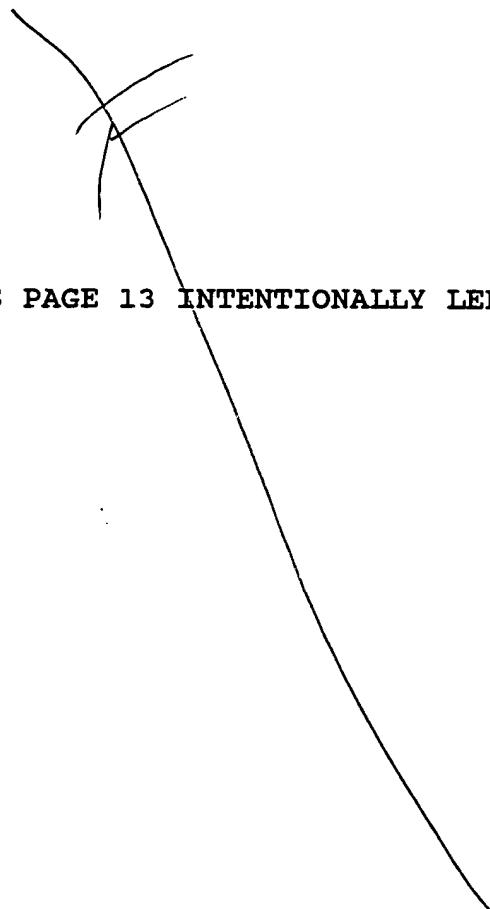
Fig 9 is a block diagram representation showing another configuration of a system for processing large database images.

To facilitate disclosure of the illustrated embodiments, the components shown in Figs. 1 to 9 of the drawings have been assigned reference numerals and a description of such components is given in the following detailed description. The components in the figures have in general been assigned reference numerals, where the hundreds digit of each reference numeral<sup>IN</sup>s corresponds to the figure number. For example, the components in Fig 1 had reference numerals between 100 and 199 and the components in Figure 2 have reference numerals between 200 and 299, except that a component appearing in successive drawing figures has maintained the first reference numeral.

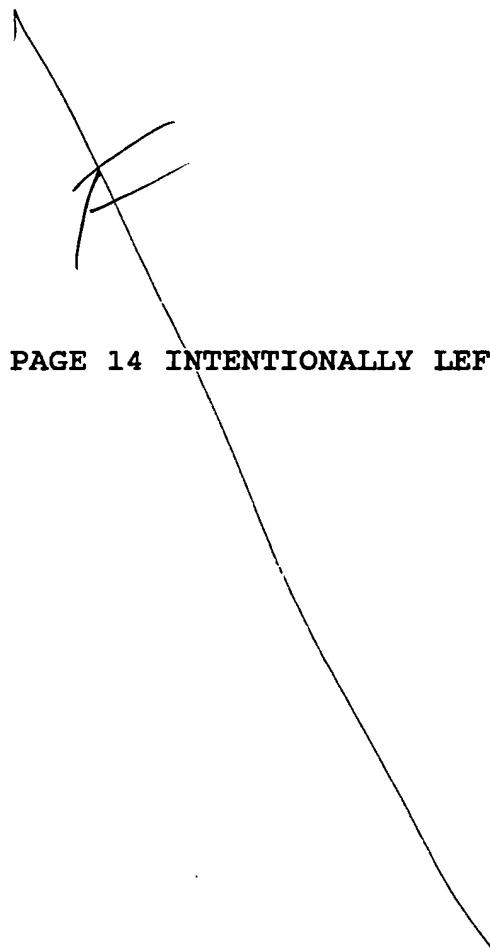


**THIS PAGE 11 INTENTIONALLY LEFT BLANK**

~~THIS PAGE 12 INTENTIONALLY LEFT BLANK~~



THIS PAGE 13 INTENTIONALLY LEFT BLANK



**THIS PAGE 14 INTENTIONALLY LEFT BLANK**

GENERAL DESCRIPTION

### General

One configuration of the system of the present invention is an image processing system capable of geometrically manipulating a highly detailed image in true real time; such as for simultaneous rotation, translation, expansion, compression, 3D perspective, and warping at a 30-times per second update rate. Other capabilities include image enhancement, smoothing, and image filtering; all in real time. The image can be obtained from a wide variety of sources; such as from a video camera or from a database memory.

The image processor can be implemented in a range of configurations, including an image processing subsystem and an image processing system. One configuration is a multiple channel high resolution image processor. Various modules are discussed to accommodate different types of input and output interfaces and multiple channel capability. The various modules of the image processor permit implementation of a basic configuration and then permit modular expansion to a multi-channel system. Modularity permits the basic configuration to be implemented in multitudes of ways using such modules.

The basic configuration with its modular flexibility can be configured into a sophisticated system with extensive capabilities, such as a turn-key image processing workstation that can be configured for various applications with appropriate software in the supervisory processor. This configuration can also include conventional interfaces between the supervisory processor and commercially available peripherals; such as a user interface comprising a computer terminal with keyboard,

joysticks, or trackball. This configuration can be implemented as a general purpose image processing workstation, computer aided instruction system, business graphics workstation, computer aided design workstation, and other turnkey systems through software adaptations in the supervisory processor.

The present invention can be used in many applications. One general purpose modular configuration is shown in Figs 1A to 1G and summarized in the MODULAR CONFIGURATION FEATURES TABLE herein. Various other configurations can also be provided.

The block diagram shown in Fig 1A illustrates the modular expandability of the system of the present invention, shown in greater detail in Figs 1B to 1G. A plurality of geometric modules 110A to 110B can be configured in parallel channel form, such as for multiple overlays. The geometrically processed images can be combined with a geometric multiplexer/demultiplexer/combiner 110D. Multiplexing selects a particular geometric processed image channel for subsequent processing. Processing includes overlaying, adding, subtracting, and otherwise selecting and combining of images. For example, many channels of geometrically processed images 110C can be overlaid with occulting priorities. Also, a pair of images can be selected for arithmetic processing, such as for adding together.

The processed and combined images can be demultiplexed with element 110D to route the appropriate images to the appropriate spatial modules 110E to 110F and for feedback to the geometric modules along path 110H. The spatial modules 110G can be

implemented in parallel form for processing the images routed thereto. A spatial multiplexer and demultiplexer 110I can be used to multiplex and demultiplex the spatially processed images from spatial modules 110G for outputting.

A plurality of input sources of images 110J and a plurality of output devices for images 110K can be accommodated. The input images from input sources 110L can be processed with the input interfaces 110M and processed with a multiplexer/demultiplexer 110N for routing to geometric modules 110C. Output devices 110K can be excited with images that are processed with the output image interfaces 110P and output multiplexer/demultiplexer 110Q.

Multiplexer/demultiplexer modules 110D, 110N, and 110Q can be implemented to multiplex a plurality of channels into one channel and then to demultiplex that one channel into a plurality of channels. Each multiplexer associated with a channel can be replicated a plurality of times to multiplex a plurality of channels into each single channel. The multiplex signals associated with the plurality of multiplexed channels can then be demultiplexed, permitting routing of any one of a plurality of multiplexer input channels to any one of a plurality of multiplexer output channels. Tri-state multiplexers can be used to multiplex a plurality of channels into a single channel, such as using 74LS365 multiplexer circuits. Parallel fanout and gating networks can be used for demultiplexing. Other types of multiplexers and demultiplexers are well-known and can be used for the multiplexer/demultiplexer modules.

A supervisory processor 110R provides supervisory operations; such as receiving external commands 110S for configuring the system. For example, geometric modules 110C can be controlled for different types of geometric processing with different geometric parameters; spatial modules 110G can be controlled for different types of weights loaded into weight RAMs; and the multiplexer/demultiplexer modules and combiner module 110I, 110N, and 110Q can be selected for multiplexing and demultiplexing of images.

The arrangement shown in Fig 1B represents one arrangement for <sup>modularly</sup> expanding the system of the present invention. However, the system of the present invention can be implemented in various subsets of this configuration and various modifications to this configuration. For example, one reduced configuration for geometric processing provides a geometric module 110A by itself, as shown in Fig 1B. For example, geometric module 110A can be used as a sub-system for other systems without the other modules shown in Fig 1A. Alternately, the system can be implemented as a single channel system as shown in Fig 1C. An image source 111A, such as a Winchester disk or floppy disk, can be used to provide images which can be processed with the input image interface 111B, such as for multiplexing and loaded into image memory 111C. Geometric processor 110D operates under control of channel processor 111E to generate geometrically processed images from image memory 111C. Spatial processor 110E spatially processes the information from geometric processor 110D to provide a spatially geometric processed image to the output image interface 111F. The output image interface 111F can

process the image, such as with video DACs, to provide appropriate signals to the output device 111G, such as providing RGB analog video signals to a color CRT monitor. Other single channel combinations can be implemented as subsets of the more general version shown in Fig 1A.

One configuration of the geometric module 111H is shown in block diagram form in Fig 1D. Image memory 111C receives image information from the input image interface for outputting to the spatial processor and to the output image interface under control of geometric processor 111D. Geometric processor 111D can rotate, translate, expand, compress, 3D process, warp, and otherwise geometrically process the image stored in image memory 111C to provide the geometrically processed output image.

Geometric processor 111C operates under control of channel processor 111D for generating initial condition information, such as under control of once per field or once per frame initial condition information. Channel processor 111E in turn operates under control of supervisory processor 110R, such as for communication of commanded motion from operator controls or from a host-computer.

One configuration of spatial module 110E is shown in block diagram form in Fig 1E. It provides spatial processing of image information. In one configuration, image information is geometrically processed and loaded into a buffer memory, such as a multiple buffered 3-line buffer memory. The 3-line buffers load kernel registers 111J for parallel generation of a 9-pixel kernel. A weight RAM 111M is loaded with weights from the

supervisory processor for the desired spatial processing. Multipliers 111K and summers 111L multiply the appropriate weights from weight RAM 111M with the corresponding intensities from kernel registers 111J to obtain weighted intensities and sum these weighted intensities to get a single weighted intensity for output to the output image interface. Sum of the products processing can provide intensity resolution enhancement and spatial resolution enhancement.

One configuration of the input interface is shown in block diagram form in Fig 1F. A plurality of sources 112A to 112F can be provided for generating image information together with input and interface circuitry for converting the input source images into a form compatible with the geometric modules. For example, analog video signals 112G can be converted to digital RGB signals 112H and multiplexed with other digital RGB signals for selection of the appropriate input image with the input multiplexer/demultiplexer 112I for image memories in the geometric modules.

Digital input images 112H can be obtained from digital database memories, digital front end sources 112J, and as feedback from other parts of the system 112K; such as from the geometric modules and spatial modules. Analog RGB signals 112L can be processed with video analog-to-digital converters (ADCs) 112M to 112N to convert the analog RGB signals 112L into digital RGB signals 112H for loading into image memories.

Composite video signals 112P can be multiplexed with multiplexer/demultiplexer 112V and converted to analog RGB signals 112Q with converters 112T to 112U, which can then be processed with the RGB video ADCs 112M and 112N for conversion to digital RGB image information 112H. The analog RGB signals can be multiplexed with multiplexer 112R for time sharing of the video ADCs. Alternately, dedicated video ADCs can be provided for parallel operation without time sharing. Analog video signals, either RGB or composite, are often provided by video tape recorders, video disks, and video cameras.

Digital image information can be provided to the geometric modules, such as for storing in image memory, or can be routed to other devices. For example, a digital frame buffer 112S can be provided for buffering input image information, such as in a double buffer arrangement, to reduce contention with image memory operations.

One configuration of an output arrangement is shown in Fig 1G. The spatial multiplexer/demultiplexer 113A can route a plurality of the spatial processor (or geometric processor) image channels to a plurality of output devices 113G to 113L under control of the supervisory processor. Digitally processed image information can be routed directly to digital devices 113K to 113L, such as a pattern recognition processor or artificial intelligence processor or a digital database memory. Digitally processed image information can also be routed to analog devices 113G to 113J through RGB video digital-to-analog converters (DACs). RGB video signals can be converted to composite video signals with converters 113E to 113F for output devices that

operate on composite signals 113I to 113J. Output devices that operate on RGB video and composite video signals include video tape recorders, video monitors, and other video output devices. Alternately, video output signals, digital or analog, can be fed-back to the input interface for recirculating through the image processor.

Video DACs 113B to 113C and RGB-to-composite converters 113E and 113F can be dedicated to output channels or can be multiplexed with multiplexers 113A and 113D respectively between a plurality of output channels under control of the supervisory processor. Auxiliary circuitry, such as a synchronization generator 113M and a blanking generator 113N, can be provided; such as for interfacing to a CRT monitor.

## MODULAR CONFIGURATION FEATURES TABLE

- I. Supervisory processor
  - A. CPU
  - B. ROM
  - C. RAM
  - D. Input/output
  - E. External commands
    - 1. Joystick
      - a. Rotation
      - b. Translation
      - c. Expand/compress
      - d. Range (perspective)
    - 2. Digital word, supervisory processor or keyboard
      - a. Rotation
        - 1) Each overlay and background
      - b. Translation
        - 1) Each overlay and background
      - c. Expand/compress
        - 1) Each overlay and background
      - d. Occulting priorities
        - 1) Each overlay
      - e. ALU processing
        - 1) Pair, overlays and/or background
      - f. Wrap-around coordinates

MODULAR CONFIGURATION FEATURES TABLE (CONTINUED)

II. Video input interface

A. Supervisory processor control

B. Input multiplexer

1. Digital RGB

- a. Geometric processed feedback
- b. ALU processed feedback
- c. Spatial processed feedback
- d. Digital image source
  - 1) Winchester disk
  - 2) External computer
    - a) VAX bus interface
    - b) Multi-bus interface

2. Analog composite video

a. Source

- 1) Video disk
- 2) Video tape
- 3) Video sensor
  - a) Vidicon
  - b) Orthocon
  - c) Infrared
  - d) Radar
  - e) Sonar

b. Converter, composite video to RGB video

c. Converter, RGB video to digital RGB

3. Analog RGB or monochrome video

a. Source

- 1) Video sensor
  - a) Vidicon
  - b) Orthocon
  - c) Infrared
  - d) Radar
  - e) Sonar

b. Converter, RGB video to digital RGB

C. Input demultiplexer

1. Supervisory processor control

2. Distribution to geometric modules

3. Frame buffer

a. Low speed input buffer

b. Higher speed double buffer

MODULAR CONFIGURATION FEATURES TABLE (CONTINUED)

III. Geometric module

A. Channel processor

1. Intel 8085 CPU
2. ROM; 16K, expandable
3. RAM; 4K, expandable
4. Input/output; 3 ports, expandable
5. Software
  - a. Pascal compiled code
  - b. Assembly language routines

B. Geometric processor

1. Rotation
  - a. Continuous
  - b. Resolution; 0.02 degrees
  - c. Range: unlimited, no hard stop
2. Compression
  - a. Continuous
  - b. Fractional (non-integer)
  - c. Resolution: 1-pixel
  - d. Range: 64-times to "hard stop"
  - e. Progressive compression
    - 2) Buffer image memory
3. Expansion
  - a. Continuous
  - b. Fractional (non-integer)
  - c. Resolution: 1-pixel
  - d. Range: 512 times to "hard stop"
4. Translation
  - a. Continuous
  - b. Resolution 1-pixel
  - c. Range: Unlimited
  - d. Hard stop consistent with database image
  - e. plus/minus X
  - f. plus/minus Y
5. 3D-Perspective
  - a. Range variable compression
  - b. Range variable intensity
  - c. Range variable motion
6. Warping
7. Mirror image
  - a. X-image reversal
  - b. Y-image reversal

MODULAR CONFIGURATION FEATURES TABLE (CONTINUED)

III. (Continued)

C. Image memory

1. 512-by-512 pixels/block
  - a. Expandable in blocks
  - b. Maximum configuration: 4096-by-4096 pixels
2. Input and output
  - a. Raster scan: 30 frames/sec max
    - 1) 13-million pixels/sec max
  - b. Random access: 13-million pixels/sec max
  - c. Sequential access, along a vector
    - 1) Start address
    - 2) Pixel words
  - d. Memory to memory transfers
  - e. Processed image to memory
3. Pixel latitude
  - a. DAC configuration is a function of option
  - b. Pixel byte: 8-bits per pixel, expandable
  - c. Color: RGB
    - 1) Bits per pixel: T (where A+B+C=T, with jumpers:
      - a) Red: A
      - b) Green: B
      - c) Blue: C
  - d. Color lookup table
    - 1) Input bits per pixel (table input): 8, expandable
    - 2) Output Bits per pixel (table output): 8, expandable
    - 3) Pallet: ROM or S/W load of RAM
  - e. Monochrome
    - 4) Bits per pixel: 8
  - f. Expandable in 8-bit per pixel bytes
    - 1) Maximum configuration: 24-bits per pixel

MODULAR CONFIGURATION FEATURES TABLE (CONTINUED)

- IV. Geometric multiplexer/demultiplexer/combiner
  - A. Supervisory processor control
  - B. Multiplexer
    - 1. Digital RGB
      - a. Geometric processed pipeline
  - C. Combiner
    - 1. Overlays
      - a. Overlay all geometric channels
      - b. Occulting
        - 1) Pixel-by-pixel occulting
        - 2) Priority
          - a) Host computer control
        - 3) Cropping
          - a) Irregular external outline
          - b) Irregular internal features
      - c. Keying
  - D. Arithmetic logic unit (ALU)
    - 1. Select two geometric channels
    - 2. Arithmetic
      - a. Add images
      - b. Subtract images
      - c. Multiple images
    - 3. Logically process images
  - E. Demultiplexer
    - 1. Digital RGB
      - a. Distribution to spatial processor modules

MODULAR CONFIGURATION FEATURES TABLE (CONTINUED)

- V. Spatial module
  - A. Parallel channels
  - B. Pipeline processor
    - 1. Buffer memory
    - 2. Kernel registers
      - a. Intensities
      - b. 9-pixel kernel, expandable to 25-pixel kernel
    - 3. Weights
      - a. RAM
        - 1) Host computer load
      - b. ROM
    - 4. Multipliers
      - a. Parallel multipliers
      - b. Intensity times weight
    - 5. Summers
      - a. Parallel adders
      - b. Sum of the products
  - C. Progressive compression
  - D. Pre-filtering
    - 1. Feedback to input mux/demux
    - 2. Low pass filter
    - 3. Anti-aliasing
  - E. Post-filtering
    - 1. Pipelined
    - 2. Spatial resolution enhancement
    - 3. Intensity resolution enhancement
      - a. Processing gain
      - b. Interpolation
    - 4. Subpixel motion enhancement
    - 5. Anti-aliasing

MODULAR CONFIGURATION FEATURES TABLE (CONTINUED)

- VI. Video output interface
  - A. Supervisory processor control
  - B. Spatial multiplexer
    - 1. Digital RGB
    - 2. Multiplexer multiple spatial channels
  - C. Spatial demultiplexer
    - 1. Digital RGB
    - 2. Demultiplexes multiple output channels
    - 3. Digital devices
      - a. Winchester disk
      - b. Pattern recognition system
      - c. Artificial intelligence system
      - d. External digital keyer
    - 4. RGB video devices
      - a. RGB video DACs
      - b. CRT monitor
      - c. External RGB video keyer
    - 5. Composite video devices
      - a. Video cassette recorder
      - b. CRT monitor
      - c. External composite keyer
      - d. RGB video DACs
      - e. RGB to composite video converter

### Experimental System

An experimental system has been configured to demonstrate many of the features of the present invention. One configuration of this experimental system will now be discussed with reference to Fig 1H. Image memory 120D is used to store an image. The image is processed under control of geometric processor 120E and scanned out for display. Geometric processor 120E can translate, rotate, compress, and expand the image from image memory 120D. The processed image is scanned out to display monitor 120H through digital to analog converter (DAC) 120G. Timing and control circuitry 120J provides timing and control signals; such as clock signals, horizontal synchronization signals, and vertical synchronization signals; for control of system operations. The supervisory processor in mainframe 120B provides an interface to database memory 120A to load images into image memory 120D. Supervisory processor 120I also controls operation of geometric processor 120E; such as by generating initial conditions to define translational position and rotational position of a display window, to define compression of the image in image memory 120D, and to interface to operator controls 120K.

For simplicity of demonstration, the database is implemented with an S-100 bus system. Disk memory 120A is implemented with dual floppy disks, mainframe 120B is implemented as an S100 bus mainframe having an 8085 microprocessor (the supervisory processor) and an 8088 microprocessor together with a RAM disk, and parallel datalink 120C to communicate database images for loading into image memory 120D. Disk memory 120A simulates a

database memory and mainframe 120B simulates a database memory interface.

As an alternate to direct scanout from geometric processor 120E, a refresh memory 120F can be used as an interface memory between geometric processor 120E and display monitor 120H. Alternately, refresh memory 120F can be eliminated, where geometric processor 120E can generate digital pixel information directly to DAC 120G for refreshing of display monitor 120H.

Operator controls 120K can include controls for X-translation, Y-translation, compression/expansion, and rotation. These controls can be implemented in different ways; such as with switches, potentiometers, encoders, and other input devices. For convenience of implementation, a pair of joysticks, each having two axis of control, were selected. One joystick implements control of X-translation and Y-translation. The other joystick implements control of rotation and compression/expansion. Operator controls 120K are interrogated by the supervisory processor operating under program control and scenario parameters are updated by the supervisory processor operating under program control in response to the operator control conditions for initializing geometric processor 120E.

Some of the more important features of the experimental system are listed in the EXPERIMENTAL CONFIGURATION FEATURES TABLE.

EXPERIMENTAL CONFIGURATION FEATURES TABLE

Real time operation  
Medium resolution (260,000-pixels per frame)  
Dynamic updates at the field rate (60-times per second)

Interactive operation  
Joysticks and computer commands  
X-axis translation  
Y-axis translation  
Rotation  
Expansion and compression  
Instantaneous response  
Simultaneous motion, all controls

Geometric processing  
Rotation  
Continuous  
Resolution: 0.2-degrees  
360-degrees/sec max rate  
Software limited  
Expandable

Translation  
Continuous  
Resolution: 1-pixel  
1000-pixels/second max rate  
Software limited  
Expandable

Expansion/compression  
Continuous  
Fractional (non-integer)  
Resolution: 1-pixel  
Double/half size per second max rate  
Software stops  
Expandable

Monitor  
Barco color monitor.  
Blanked to 484-lines and about 700-pixels/line  
3-colors, RGB  
13-inch CRT  
Inline gun shadow mask

Image memory  
512-pixels by 512-pixels  
7-bits per pixel  
2-bits red  
2-bits blue  
3-bits green  
200-ns RAMs

Spatial filtering  
9-pixel kernel at 10-million kernels/sec  
Weight RAM

Virtual scrolling  
Image memory wrap-around

Supervisory computer  
8085 8-bit microcomputer  
CP/M-80 operating system  
Compiled Basic

The features of the present invention can be implemented in various ways. Various configurations have been discussed, such as with reference to Figs 1A to 1H, 34A, 35D. Another configuration will now be discussed with reference to Fig 1I.

Database memory 130A can be a digital database memory for storing digital image information to be loaded into buffer memory 130B. Preprocessor 130C can be used to perform various types of preprocessing; such as low pass filtering, shading, topological line generation, and pseudo-color generation. Preprocessing can be performed slower than real time, where real time operation can be performed with geometric processor 111D and where preprocessed information may be needed slower than real time to load image memory 111C. The image in buffer memory 130B can be low pass filtered and shaded and can have topological lines drawn and pseudo-color applied prior to loading into image memory 111C.

Low pass filtering can be implemented with kernel processing, such as discussed for the FTR.ASC program set forth in the BASIC PROGRAM LISTING FTR.ASC herein and as discussed for the spatial filter herein. The weight kernel can be shaded, such as with cosine shading weights, or can be unshaded, such as with constant weights.

The image in image memory 111C can be loaded from buffer memory 130B in response to geometric processing; such as in response to translation, expansion, and compression processing. The image in image memory 111C can be processed under control of geometric processor 111D; such as for rotation, translation, expansion, compression, 3D-perspective, warping, windowing, and overlaying. Some of the geometric processor operations are

relatively slow operations that can be performed with a serial processor, such as serial processor 130D; as discussed below.

In one configuration, the geometric processor can be implemented having parallel arithmetic operations for generating pixel addresses along a line and for generating line addresses. The pixel addresses along the line are typically generated at relatively high bandwidth, such as at 10-MHz. The row addresses are typically generated at relatively low bandwidth, such as at 15-KHz. Consequently, the row addresses and other row related processing can be implemented in time shared form, such as with serial processor 130D in combination with the pixel addresses that are generated at relatively high bandwidth with a parallel processor. A serial processor can be implemented to time share arithmetic units associated with the row address generators and to time share other processing; such as 3D-perspective processing for changing row-related parameters. Serial processor 130D can be implemented with a difference equation processor, such as a digital differential analyzer (DDA) as discussed in Patent No. 3,586,837; which is incorporated herein by reference. A serial DDA can be implemented with a computational element time shared between a plurality of computational operations that are stored in a memory for time sharing the computational element.

Occulting can be implemented with occulting processor 130K and occulting buffers 130L and 130M, such as for topological occulting. As previously discussed, geometric processor 111D can geometrically process the image in image memory 111C to scanout a geometrically processed image, which can be loaded into

occulting buffer 130L as a geometrically processed rectilinear coordinate image. Occulting processor 130K can process the rectilinear image in occulting buffer 130L in polar coordinate form, radially scanning the buffered image to introduce occulting projections as a function of pixel altitude and scanning or projecting the image into occulting buffer 130M in polar coordinate occulted form. Occulting buffer 130M can be considered to perform a polar coordinate to rectilinear coordinate conversion by scanning out the occulted image in raster scan form.

A sensor input can be provided, such as from camera 130G; generating real time imagery for displaying with the database imagery. The camera signals can be processed with converter 130H, such as for NTSC to analog RGB and analog RGB to digital RGB conversion. The digital RGB information can be loaded into an image buffer 130I. With image buffer 130I implemented in a double buffer configuration, a new image can be loaded into an input buffer simultaneously with a prior image being scanned out for display. With image buffer 130I implemented in a single buffer configuration, a new image can be loaded into a single buffer simultaneously with a prior image being scanned out for display. The real time camera image in buffer 130I can be geometrically processed with geometric processor 130E to superimpose dynamic real time geometric processing on dynamic real time camera images. The images in image memory 111C and camera buffer memory 130I can be overlaid or can be windowed together, such as with multiplexer 130N under control of geometric processor 111D. For overlaying, geometric processor

111D can be implemented as a multiple geometric processor for simultaneous geometric processing of the 2-images for overlaying therebetween. For windowing, geometric processor 111D can be implemented as a single geometric processor time shared between the 2-images for display in windows of the viewport.

CRT interface 130P can be implemented to interface the digital pixel information with the CRT monitor. It can include a synchronization generator, blanking circuit, and digital to analog converters.

Range variable intensity can be implemented with a multiplying DAC in the CRT interface 130P controlled by a range parameter, such as generated in geometric processor 111D. The range parameter can be generated in conjunction with 3D-perspective geometric processing or can be generated with other processing. The range parameter generated in digital form can be applied to a multiplying DAC to generate an analog voltage that is a function of range. The analog voltage can be applied to the reference of the video DACs for modulating image intensity as a function of range.

Other elements can be included in the arrangement, such as a spatial processor operating at sub-pixel resolution.

### Discussion Of Figs 1J And 1K

An alternate configuration will now be discussed with reference to Fig 1J. Supervisory processor 131F can control system operation. It can provide interactive communication with an operator through operator panel 131I and with host computer 131H through data link 131G. Supervisory processor 131H can control a simulation scenario under control of scenario driving functions generated by an operator and by a host computer. High detail information, such as terrain information, can be stored in database memory 131B. A hierarchial memory map architecture can be used; comprising database memory 131B, image memory 131D, and refresh memory 131K. Database memory 131B can be implemented to contain large amounts of high detail information in memory map form, which can be selectively accessed for loading into image memory 131D. Image memory 131D can be implemented in memory map form for an area of the environment that is larger than the viewport area. Refresh memory 131K can be implemented in memory map form and clipped to the viewport for display on a CRT monitor.

The hierarchial memory map architecture extends to memory traffic. Database memory traffic may be lowest, being used to update image memory 131D as a function of scenario progression between mosaic frames for motion past frames. Image memory traffic may be higher, being used to update refresh memory 131K as a function scenario progression within a mosaic frame for motion past pixels. Refresh memory traffic can be highest, being used to refresh monitor 131L at about a 10-MHz pixel rate for medium resolution display (512 lines) and at about a 40-MHz pixel

rate for a high resolution display (1024 lines).

Image processor 131E can access static area information from image memory 131D to transform this static information to real time dynamic geometric information for loading into refresh memory 131L. Image processor 131F can translate, rotate, warp, spatially compress, and otherwise transform the static area information in order to provide the appearance of dynamic motion.

Database memory 131B can store large amounts of high detail information, such as textured terrain information. This information can be stored on a video disk in analog form as video frames of information representing a mosaic of images. Frames can be selected by supervisory processor 131F, such as in real time under control of a dynamic allocation program using a lookahead for frames of information to be loaded into image memory 131D. Database memory 131B can be used in conjunction with an analog-to-digital converter (ADC) 131C, such as a flash ADC, to provide digital frame information for storage in image memory 131D.

A video disk can store about 50,000 mosaic frames (about 10-billion pixels) per side. The frames can be stored having the highest spatial resolution that is required, then can be spatially compressed with image processor 131E to form an image suitable for the particular display scenario. For example, in a flight simulator application, frames can be stored for the highest zoom magnification and lowest aircraft altitude required, then can be spatially compressed to the lower zoom magnification and the higher aircraft altitude as they vary through the dynamic

simulation scenario.

Image memory 131D can store multiple frames of information in mosaic form. A viewport window can be controlled by image processor 131D for selecting pixels to be displayed. Image processor 131D can perform transform processing in real time to display the pixels in the viewport having appropriate dynamic conditions. The transformed pixels in the viewport can be loaded into refresh memory 131K and used to refresh CRT monitor 131L. Image memory 131D can be configured as a mosaic of video frames, as shown in Fig 1K. The viewport is implied by the crosshatched window 131M, which can be rotated, translated, and otherwise geometrically processed in image memory 131D by image processor 131E. As viewport 131M translates towards an edge of image memory 131D, supervisory processor 131F identifies the frames to be accessed from database memory 131B for extending that edge of image memory 131D. Image memory 131D can be implemented in a wrap-around form, where adding of mosaic frames to an edge of image memory 131D extends that edge and overwrites the opposite edge. This has the effect of keeping viewport 131M near the center of image memory 131D. Wrap-around capability can be provided in all four coordinate directions, where image memory 131D may be considered to be spherical in nature. This permits viewport 131M to continuously move in a particular direction preceded by new mosaic elements selected for progressively extending image memory 131D in the direction of motion.

Image processor 131E provides real time rotation, translation, spatial compression, anti-aliasing, and other image processing operations to achieve a dynamic image derived from the

static image in image memory 131D under control of the driving functions generated by supervisory processor 131F. Image processor 131E also provides the capability to warp the image; such as to warp for simulation applications or to remove warp for display precision. Spatial compression can be implemented as a function of range to provide a 3D distance perspective. Viewport 131M can be panned over image memory 131D as a function of tilt or aircraft bank angle in conjunction with range-related spatial compression to suitably compressed an image as a function of range. Such panning and spatial compression can be along lines of constant slant range, which necessarily crosses lines and columns of image memory 131D at angles consistent with aircraft roll, pitch, and heading angles. Consequently, spatial compression can represent the superposition of various parameters; such as slant range, altitude, zoom, and other related parameters.

Image processor 131E can be implemented in various alternate configurations. In one configuration, image processor 131 can be implemented in digital form. Alternately, analog and hybrid configurations thereof can be provided.

In a digital image processor configuration, digital processing can be implemented in special purpose or general purpose forms. Special purpose logic, such as digital differential analyzers (DDAs), can provide rotation, translation, spatial compression, and other processing. General purpose processors, such as a stored program computer and a microcomputer, can be used for a software implementation and a

firmware implementation respectively. For example, an AMD 2900 bit-slice microprocessor can be used for a firmware implementation. Alternately, such processing can be performed with special purpose logic. Processing can take various forms, such as 2-dimensional matrix transformations and incremental processing. Matrix transformations can include a group of two dimensional rotation, translation, scaling, and compression matrices.

In an analog configuration, analog signals can be generated relating to translation, rotation, and compression parameters and can be manipulated with analog circuits; such as analog differential amplifiers for addition and subtraction, analog nonlinear processors for multiplication and division, and analog function generators.

In a hybrid configuration, parameters can be generated in analog and digital form and can be processed with hybrid processing circuits, such as multiplying digital-to-analog converters (DACs). For example, addition and subtraction can be performed in the analog domain, such as with differential amplifiers, and nonlinear functions, such as multiplication, can be performed in the hybrid domain, such as with multiplying DACs.

Initial conditions and intermediate parameters can be generated with supervisory processor 131F to support image processing operations. For example, initial conditions can be generated for incremental elements, such as the arc center parameters and endpoint parameters for a DDA circle generator and slope parameters and endpoint parameters for a DDA vector generator.

The consideration of parallel accessing of multiple pixels from image memory 131D in the presence of rotation will now be discussed. A configuration can be provided to simultaneously access multiple adjacent pixels from image memory for spatial filtering (with weighting) into a single output pixel. Image memory 131D can be partitioned into planes and/or blocks for parallel accessing of multiple pixels. Rotation need not cut across boundaries of memory planes and/or blocks relative to parallel accessing of pixels. This is because adjacency between pixels can be fixed and need not vary as a function of rotation. In one configuration, the processing selects the center pixel of a group of pixels in image memory 131D, accesses the selected pixel and adjacent pixels from image memory in parallel, weights the accessed pixels in parallel, sums the weighted pixels, and transfers the resultant spatially filtered pixel to the appropriate rotated pixel address in refresh memory 131K. The relationship between the address of the center pixel in image memory and the address of the spatially filtered pixel in refresh memory 131K changes as a function of rotation. However, the pixels adjacent to the selected center pixel in image memory do not change as a function of rotation. Therefore, the above described parallel accessing, weighting, and summing of adjacent pixels from image memory 131D need not change as a function of image rotation.

The configuration discussed with reference to Fig 1J can be implemented in the form of a single terminal or multiple terminal configuration. In a multiple terminal configuration, resources

can be shared between terminals, such as to handle peak loads. This can reduce costs and increase capabilities. Costs can be reduced because each terminal needs the dedicated processing resources to handle the average processing load, but does not need the additional dedicated processing resources to handle the peak processing load. Capabilities are increased because a global peak load processor shared between multiple terminals can have significantly greater processing capability, such as based upon cost constraints, than a dedicated peak load processor at each terminal.

Use of a global front end can reduce costs and increase capabilities. Costs can be reduced by providing a single front end shared by multiple terminals. Capabilities can be increased because a global front end shared between multiple terminals can have significantly greater capability, such as based upon cost constraints, than a dedicated front end for each terminal.

Zoom capability provides a good example of advantages of a global peak-load processor. Zoom may be a peak load task because of the need to rapidly compress or decompress an image by a large factor. This can involve integrating and anti-aliasing of a large number of pixels in a short period. However, the duty cycle for zoom processing may be very low, where contention between two terminals generating zoom commands simultaneously may be low. Therefore, a single zoom processor can be shared between all terminals, providing significant peak load capability yet having a low cost per terminal i.e.; one sixteenth of the zoom processor cost assessed against each terminal in a sixteen terminal system.

In addition to peak load considerations discussed above, other considerations can be optimized for a multiple terminal system. For example, if each of a group of terminals is constrained to the same general display scenario; then database memory 131B, ADC 131C, and image memory 131D can be implemented as a global front end and can be shared between all terminals in the group. Use of a dedicated image processor 131E and dedicated refresh memory 131K for each terminal in combination with the global front end permits independent operation of each terminal.

Image memory 131D provides a good example of advantages of a global front end. Image memory size is an important cost and performance consideration. A larger image memory reduces traffic between database memory 131B and image memory 131D and reduces lookahead processing in supervisory processor 131F. Sharing of image memory 131D between a plurality of terminals can permit a larger image database to be implemented for reduced database traffic and supervisory processor loading.

The system discussed above with reference to Fig 1J can be provided in various alternate configurations. For example, the system can be implemented as a stand alone system without an external host computer 131H or alternately with the host computer functions being performed internally, such as in supervisory processor 131F. Database memory 131B, discussed above in the configuration of an analog video disk, can be implemented with the other analog memory devices. For example, an analog video tape can be used as an alternate to an analog video disk. Alternately, database memory 131B can be implemented with an

analog CCD memory; such as discussed in U.S. Patent No. 4,209,853; 4,209,852; No. 4,209,843; and 4,322,819 and U.S. Patent applications S/N 812,285, S/N 844,765; and S/N 160,871; which are herein incorporated by reference. *INS G*

Alternately, database memory 131B can be implemented with digital memories; such as digital video disk memories, digital magnetic disk memories, digital integrated circuit memories, digital magnetic tape, and other digital memories. For example, a large digital magnetic disk memory having about 400-megabytes of storage is available from Fujitsu and can be used for a digital database memory. Photographic and optical memories can be used for database memory 131B. For example, the environment can be recorded on frames of film which can be selected and scanned with a video camera to generate database information.

Image memory 131D and refresh memory 131K have been discussed above in the form of IC-RAM based memories.

Alternately, image memory 131D and refresh memory 131K can be implemented with other memory devices. For example, image memory 131D and refresh memory 131K can be implemented with CCD memories, such as described in the above-referenced patents and patent applications. The CCD memory may be an analog or a digital CCD memory. For example, an analog CCD memory may be used in conjunction with an analog database memory for storing analog information in an image memory that is accessed from an analog database memory without an interfacing ADC. In this configuration, analog refresh information can be loaded from the analog image memory and stored in an analog refresh memory.

Image processor 131E can be a digital processor, or alternately can be a hybrid (analog and digital) or analog processor. Hybrid and analog processors can be readily interfaced to an analog memory, such as an analog image memory and an analog refresh memory, for analog or hybrid transform processing. Various forms of analog and hybrid processing are discussed in the above referenced CCD patents and patent applications.

The configuration discussed with reference to Fig 1J above is shown with a 3-tier memory hierarchy. Alternately, other hierarchial arrangements can be provided. For example, image memory 131D can be supplemented with a fourth memory tier, a dynamic memory map; where the static image stored in image memory can be converted to a dynamic image and loaded into the dynamic memory map. For example, the static memory map in image memory can have a mosaic of the images accessed from database memory and the dynamic memory map can have this mosaic of images converted to dynamic form as a function of rotation, translation, spatial compression, and other updates by image processor 131E. The viewport can be positioned within the dynamic memory map, which can be a dynamically updated version of the static memory map (Fig 1K). As the dynamic conditions change, the dynamic memory map can be regenerated from the static memory map in image memory. This provides the advantages of having an image memory that has not been modified for dynamic conditions, such as with the database memory information, and that has rapid access (more rapid than database memory access) for restructuring the dynamic memory map as a function of scenario dynamics. The dynamic

memory can be smaller than the image memory and larger than the refresh memory. For example, the dynamic memory can be smaller than the image memory because the dynamic memory has faster access to the image in the image memory than the image memory has to the image in the database memory. The dynamic memory can be larger than the refresh memory to provide a multiple frame mosaic. The dynamic memory and refresh memory combination can be implemented with a double buffered refresh memory, where an old image is being used to refresh the display while a new image is being transformed from the image memory.

### Discussion of Figs 1L To 10

An arrangement will now be discussed with reference to Figs 1L to 10 for image processing having rotation, translation, and compression of an image to further illustrate various features of the present invention. The arrangement shown in Fig 1L can be related to the arrangement discussed with reference to Fig 1J as follows. Input device 132A may be database memory 131B, input memory 132B may be image memory 132D, image processor 132C may be image processor 131E, output memory 132D may be refresh memory 132K, and display monitor 132E may be monitor 131L.

One form of this arrangement will be characterized as transferring memory map information from a first memory to a second memory and transforming the nature of the memory map information; such as with translation, rotation, and spatial compression; during the transfer. Edge processing may be performed by selectively accessing pixels from the input memory map and selectively storing these pixels in the output memory map; where the selection of the input and output pixels facilitates image processing.

An input device 132A generates an input signal to an input memory 132B for storage in memory map form. Image processor 132C processes the input memory map stored in input memory 132B to generate an output memory map for storage in output memory 132D. Alternately, input signals 132M may be processed directly with image processor 132C without use of input memory 132B. The memory map in output memory 132D can be used to refresh display monitor 132E for display of a processed image. Input device 132A may be a data acquisition system such as a radar, sonar, video, or other

system for acquiring an image; a database memory; or other input device. The image can be stored in memory map form in input memory 132B for processing with image processor 132C. Input signals 132M may be scan-related signals; such as raster scan signals for a video input, PPI or polar coordinate scan signals for sonar and radar systems, or other scan-related signals. Alternately, input signals 132M may be obtained from a host computer, a database, or other source of image information.

Two memories, input memory 132B and output memory 132D, are shown for storing memory map information. In one configuration, image processor 132C accesses memory map information from input memory 132B for processing and subsequent storage in output memory 132D in memory map form. Alternately, the input memory map and the output memory map may be stored in the same memory, such as for in-place processing; may be stored in other than memory map form; or may otherwise be stored.

Image processor 132C may include line generators, such as discussed with reference to Fig 10 herein. The line generators can be used to map pixels from an input memory map into an output memory map. Such mapping may include selection of an array of pixel words from an input memory map and storing the selected array of pixel words in an output memory map. Transferring of arrays in this manner can be used to provide translation, rotation, zooming, panning, and other such operations.

Address generators can be used to map an input rectangular array of pixels into an output rectangular array of pixels, such as for translation. They can be used to map an input rectangular

array of pixels into an output non-rectangular array of pixels, such as for rotation and translation. They can be used to map an input non-rectangular array of pixels into an output rectangular array of pixels, such as for rotation and translation. They can be used to map an input non-rectangular array of pixels into an output rectangular array of pixels, such as for rotation and translation. They can be used to map an input non-rectangular array of pixels into an output non-rectangular array of pixels, such as for rotation and translation. Similarly, mapping of one array of pixels into another array of pixels can be used to transform coordinates, such as polar coordinates to rectilinear coordinates and rectilinear coordinates to polar coordinates, and can be used to perform other transformation and image processing operations.

Address generator operation will now be discussed with reference to Fig 1M. The image processor can use one or more address generators. For example, a pair of address generators (a pixel address generator and a row startpoint address generator) may be used to address arrays of pixels 132F to 132J, and 132N. A pixel address generator can begin at the startpoint of the line, such as at the left hand pixel of the line, and can increment across the line at the selected slope. The selected slope is zero for array 132F and is 0.5 for arrays 132J and 132N. The startpoint or row address generator selects the startpoint of the line. It is incremented at the completion of a line, generated with the pixel address generator, to generate the

selected slope of the startpoints 132H. This selected slope is zero for arrays 132F and 132J and 0.5 for array 132N. Operation can proceed by transferring a startpoint coordinate in the startpoint row address generator to the pixel address generator, generating the coordinates of the pixels along the line with the pixel address generator, and clocking the startpoint row address generator at the end of the increment to advance the startpoint row address generator to the next startpoint pixel coordinate of the next subsequent row. This next startpoint pixel coordinate is transferred to the pixel address generator as the initial condition for the next scanline. The pixel address generator is clocked to generate the coordinates of the pixels along the next scanline. A distance-to-go (DTG) or endpoint calculation may be used for each pixel address processor. The last pixel in each line can be identified with such a DTG endpoint implementation. Similarly, the last line of the array can be defined with such a DTG or endpoint implementation in the startpoint, row address generator. Simultaneous generation of a last pixel per line condition and a last line condition can identify the completion of the frame scanout.

Each memory, input memory 132B and output memory 132D, can have corresponding sets of address generators. The input address generators associated with input memory 132B can be used to address an array of pixels from input memory 132B for accessing. The output address generators associated with output memory 132D can be used to address an array of pixels in output memory 132D for storing. Therefore, separate arrays in input memory 132B and

output memory 132D can be independently addressed by separate address generators. In this way, an input array of pixels and an output array of pixels may be independently and separately identified and addressed in a pixel-by-pixel fashion to map selected pixels from input memory 132B into selected pixels in output memory 132D.

Spatial compression and decompression can be performed by undersampling an input array in input memory 132B or an output array in output memory 132D. For example, the address generators for the input array can be multiple clocked; such as double clocked, triple clocked, or quadruple clocked; for undersampling of an input array for spatial compression when loading input memory 132B into output memory 132D. Alternately, the address generators for the output array can be multiple clocked; such as double clocked, triple clocked, or quadruple clocked for spatial decompression when loading input memory 132B into output memory 132D. Integration and weighting of a plurality of pixels can be performed to reduce effects such as aliasing and windowing. Such integration and weighting can be performed as each subsequent pixel is scanned. Weighting and integration techniques are discussed for a process-on-the-fly arrangement in patent No. 4,209,843; which is herein incorporated by reference. For example, the input samples T therein may be a scanned pixel array stored in input memory 132B herein, the weighting may be performed with elements 625 and 626 therein, and the output memory 614 therein may be output memory 132D herein for accessing and integrating weighted samples (Fig 6D therein and Fig 1L herein).

An aircraft map following navigation application will now be discussed to illustrate use of the arrangement discussed with reference to Fig 1L. This application presents a dynamic map display from a pre-mapped recorded database for aiding in navigation. An image of the terrain below an airplane is stored in memory map form in input memory 132B. This can be a prerecorded image stored in database and loaded into input memory 132B. The database may be self contained in the system or may be transmitted from a remote source. Input device 132A can be a map database generating map video signals 132M for storage in input memory 132B in memory mapped form. Loading of signals 132M into input memory 132C can be performed with address generators identifying the pixels along the scan line for storing of scan line information or with various other methods for loading a memory map with image information. The memory map in input memory 132B may be a composite of many map images or mosaics over a larger area. Arrays in the input memory map in input memory 132B may be selected and processed with image processor 132C for loading into output memory 132D. As the aircraft translates over the terrain, the startpoint pixel of the array is translated over the input memory map in input memory 132B to translate the viewing window displayed on display monitor 132E consistent with aircraft motion. Similarly, as the aircraft rotates in azimuth, the address generators are loaded with the slopes of the lines of the array, thereby effectively rotating the displayed window in the memory map stored in input memory 132B. Therefore, as the aircraft translates and rotates over the terrain, the display

window similarly translates and rotates over the memory map to present an image depicting the environment below the plane.

The azimuth angle of the aircraft can be used as a driving function for the address generators. The heading angle can be processed with an arc tangent calculation to obtain the slope of the line and the component rectilinear vectors. Alternate address generator embodiments, such as using either slope or rectilinear component vectors, can be provided. Therefore, the initial conditions for the address generator can be derived from the azimuth angle for rotation.

Overlays may be provided with the arrangement shown in Fig 1L. For example, overlays including vectors, points, alphanumerics, and symbols can be generated in input memory 132B in memory map form. These overlays can be loaded destructively over the processed image, effectively erasing the information contained thereunder, or nondestructive by loading output memory map bit planes for overlays. The memory mapped overlays correspond to the memory mapped image and therefore can be translated, rotated, and otherwise processed with the image. Therefore, overlays may be processed in similar manner to and in conjunction with the images discussed above. Alternately, overlays can be rotated, translated, and otherwise processed, such as with a geometric processor in line endpoint form and then generated with vector generators and character generators for introduction into output memory 132D.

Various arrays of pixels will now be discussed with reference to Fig 1M. A rectangular array of pixels 132F contains horizontal rows and vertical columns of pixels organized in a

rectilinear coordinate system. A non-rectangular array of pixels 132J contains lines of pixels crossing rectilinear lines at an angle. Array 132J has lines in the rectilinear coordinate system and lines not in the rectilinear coordinate system. Array 132N has no line in the rectilinear coordinate system. Each line of array 132F can be scanned with an address generator having a slope of zero and each line of arrays 132J and 132N can be scanned with an address generator having a slope of 1/2.

Subsequent lines progressing downward in each of the arrays 132F, 132J, and 132N can be scanned by decrementing (or incrementing) the startpoint pixel address of the prior scanline to address the startpoint pixel for the next subsequent scanline. For example, the startpoint pixel for arrays 132F and 132J can be decremented in Y and the startpoint for array 132N can be decremented in Y and partially decremented in X to arrive at the startpoint pixel for the next subsequent scanline. Decrementing in rectilinear coordinates can be provided with address generators. Therefore, array 132F can be generated by incrementing the X-pixel address generator to generate a rectilinear line of pixels and then decrementing the Y-pixel address generator to address the next scanline of pixel addresses. Array 132J can be generated by updating pixel addresses along a scanline in accordance with the line slope, such as implemented with an address generator, and then decrementing the Y-pixel address register to address the next line of pixel addresses. Array 132N can be generated by updating pixel addresses along a scanline in accordance with the line slope and then addressing the next line of pixels and

updating the pixel address generators with a startpoint row address generator using a second address generator along another slope. In this way, address generators can be used to load one array of pixels into another array of pixels.

Spatial compression can be implemented by undersampling, such as selecting every second pixel, every third pixel, every fourth pixel, etc along the scanline and similarly selecting equal steps in the scanline for mapping a smaller array into a larger array for spatial expansion and mapping a larger array into a smaller array for spatial compression. Such spatial expansion and compression can be performed along rectilinear coordinates or non-rectilinear coordinates, such as discussed with reference to Fig 1M above.

In alternate configurations, the input memory map in memory 132B can be larger than, equal to, or smaller than the output memory map in memory 132D. In a configuration where the input memory map is larger than the output memory map, the output image can roam through the input memory map; such as by transferring a portion of the input memory map; properly rotated, translated, scaled, and otherwise processed; into the output memory map for displaying of a rotated, translated, scaled and otherwise processed portion of the image in the memory map stored in input memory 132B on display monitor 132E. For example, input memory 132B may store a large map of an environment (either sonar, radar, video, or otherwise) and output memory 132D may store a portion of the environment pertinent to the operation in progress, such as the portion of the environment around a vehicle being navigated. The array selected from the large memory map

stored in input memory 132B can be translated and rotated consistent with the position and orientation of a vehicle for storing as a rotated and translated portion of the image in output memory 132D.

A translation example will now be discussed with reference to Fig 1N as illustrative of rotation and other processing. A larger memory map is illustrated as stored in input memory 132B with pixel array 132P having a visual event 132Q contained therein. A first array of pixels 132R is selected and transferred to output memory 132D in memory map form having event 132Q contained therein. For subsequent processing, the selected array is translated in X and Y to array position 132S having event 132Q in the upper left hand corner. This is an apparent translation of the image to the lower right, from array 132R to array 132S within input array 132P and an apparent translation of event 132Q towards the upper left of the output arrays 132R and 132S. Output arrays 132R and 132S are shown superimposed on input array 132P and are also shown separately to illustrate the translation of output arrays 132R and 132S relative to input array 132P and to illustrate the apparent motion of event 132Q from the near center position in output array 132R to the upper left position in output array 132S. Similarly, image rotation, compression, decompression, and other image processing can be performed by manipulating memory maps.

An image processing arrangement in accordance with Figs 1L to 1N will now be discussed relative to the flow and state diagram of Fig 10. Two sets of address generators will be used.

The first set generates addresses for accessing a selected array of pixels from input memory 132B. The second set generates addresses for storing the accessed pixels in an array in output memory 132D. Each set has two address generators, a pixel address generator for generating a scanline of pixel addresses beginning at a startpoint and a startpoint row address generator for generating the startpoint pixel addresses.

Initial conditions are generated for the address generators in operation 133A. Initial conditions can include the startpoint pixel address for the array, the delta-X and delta-Y components for the slope of each line of pixels for each address generator, the X-DTG and Y-DTG parameters for each address generator, the delta-X and delta-Y parameters for the slope of the startpoints of the scanlines of pixels for each startpoint row address generator, and the X-DTG and Y-DTG parameters for each startpoint row address generator.

The X-DTG and Y-DTG parameters for the address generator are loaded in operation 133B. For an array having equal length lines, such as shown in Fig 1M, the X-DTG and Y-DTG parameters are the same for each line. In other configurations, these parameters may be varied to provide different types of arrays.

The pixel address generators are clocked in operation 133C to generate the new output conditions. The pixel address generators for both, input memory 132B and output memory 132D, are clocked in operation 133C to advance to the next pixel along the input line and output line of pixels. The output conditions are generated, such as with a table lookup, and the X-DTG and Y-DTG parameters are decremented in operation 133D. In operation

133E, a pixel word is accessed from input memory 132B, as addressed by the input pixel address generator, and stored in output memory 132D, as addressed by the output pixel address generator. A test is then made in operation 133F to detect the last pixel in the line. If it is not the last pixel in the line, the logic loops back along the NO path to again clock the pixel address generators, in operation 133C to process the next pixel in the line. When the last pixel in the line is detected in operation 133F by detection of the DTG parameters being at zero, the logic branches along the YES path to operation 133G to detect the last line. If the last line is not present, the logic branches along the NO path to initialize processing of the next line of pixels with operations 133H and 133I. If the last line is present, the logic branches along the YES path to exit the iterative loops as completing the transfer of the selected array of pixels.

A new line is initialized by clocking the startpoint row address generator in operation 133H and loading the new startpoint initial conditions generated with the startpoint row address generator into the pixel address generator for each memory to initiate transfer of the next line of pixels. The logic then loops back to operation 133B to load the X-DTG and Y-DTG parameters for the next line of pixels and then generates the next line with operations 133C to 133F as discussed above.

Address generator operations discussed with reference to Fig 10 provides for iteratively scanning an input array in input memory 132B and an output array in output memory 132D for

transferring pixel words from the input array to the output array. As discussed herein; the input and output arrays may have translation, rotation, scaling, and other differences therebetween to perform such operations on the image displayed with monitor 132E.

### Other Configurations

Various configurations of the system of the present invention have been discussed with reference to Fig 1 above to illustrate how the various features and devices of the system of the present invention can be used together to implement a system. These configurations are illustrative of the very large number of other configurations that can be implemented from the teachings herein. The configurations discussed herein and the large number of other configurations that can be implemented with the teachings herein can be used in different applications; exemplary applications being disclosed in the section related to applications herein.

The display system of the present invention has been discussed for operation in conjunction with a raster scan RGB color CRT display monitor. Alternately, the system of the present invention can be used with scan arrangements other than raster scan arrangements, monitors other than CRT display monitors, color arrangements other than RGB color arrangements, and non-color monochromatic display monitors and can be used in systems without displaying the processed images. For example, other electron beam control arrangements; such as PPI scan, A-scan, calligraphic, Palmer scan, and other arrangements can be used. For example, the geometric processor can scan the image out of image memory along the scanlines other than raster scanlines; or the geometric processor can scan the image out of image memory and into a refresh memory, where the desired scan is used to scan the image out of the refresh memory. Also, the display monitor can be implemented with a calligraphic CRT display monitor, a

liquid crystal display monitor, a plasma display monitor, and other display monitors.

The different features disclosed herein can be used in different combinations and with different interconnections, illustrated by the exemplary combinations and interconnections discussed herein but not limited to the exemplary combinations and interconnections discussed herein.

The block diagrams shown herein represent various configurations in accordance with the teachings of the present invention. However, many other configurations can be implemented with the teachings of the present <sup>invention</sup> ~~invention~~. For example, various elements have been shown separately for convenience of discussion, where such elements can be integrated together. For example, the geometric processor and spatial processor, the preprocessor and the geometric processor, and other elements can be integrated together into combined elements. Also, various elements have been shown integrated together for convenience of discussion, where such elements can be implemented separate from each other. For example, the geometric processor has been discussed for implementing the combined capabilities of rotation, translation, expansion, compression, 3D-perspective, warping, overlaying, and other functions; where such capabilities can be implemented separate from each other or in various combinations of being implemented together and separately from each other.

The input sources of image information and the output destinations of image information have been discussed in general and specific examples of input sources and output destinations

F have been discussed. Many different input sources and output destinations can be provided. Input <sup>or</sup> <sub>A</sub> sources can be image sensors, image generators, memories, other image processing systems, digitizers, communication links, and multitudes of other input sources. Output destinations can be display monitors, pattern recognition devices, artificial intelligence devices, memories, other image processing systems, communication links, and multitudes of other output destinations.

Various data paths are discussed herein to illustrate exemplary configurations. Many other data path configurations can be implemented. For example, pixel data streams can be fed-forward to downstream processing elements, fed-back to upstream processing elements, buffered, processed, input, output, and otherwise routed to various processors, memories, interfaces, and other elements.

The disclosures herein are primarily related to an image processing system. However, many of the teachings herein are not limited to image processing systems. For example, the image memory architecture and buffer memory architecture teachings herein are generally applicable; such as to data processing systems, signal processing systems, graphic <sup>ys</sup> <sub>A</sub> systems, and other systems. The filtering teachings herein are generally applicable, such as for spatial and temporal filtering of radar, seismic, and sonar systems; kernel processing of information; multiplier and sum of the products architectures for computations; table processing as with the weight table architecture; and latching of multiple parameters in multi-parameter registers for parallel processing. The system architectural teachings herein are

generally applicable, such as for pipelining of processors and memories; buffering of information; and parallel processing of information.

Various forms of buffering of information is disclosed herein; such as buffering in intermediate buffer memories and buffering in registers, i.e., kernel registers. The system arrangement discussed herein provides for multiple processors operating in parallel form and pipeline form. The parallel processing form is exemplified with the multiple geometric processor channels. The pipeline processing form is exemplified with the preprocessor, geometric processor, and spatial processor pipeline. Various buffer memories can be inserted between processors and in conjunction with processors as needed. For example, image memories are used in conjunction with geometric processors and buffer memories are used in conjunction with the preprocessor. Also, buffer memories can be inserted between the geometric processor and the spatial processor, such as the triple line buffer memory, and between the spatial processor and the output destination device, such as the refresh memory. Such memories can be inserted and deleted as needed in accordance with the teachings of the present <sup>invention</sup>. For example, a refresh memory can be inserted or deleted, as disclosed in various configurations; image memories can be inserted or deleted, as disclosed in various configurations; and various buffer memories and other memories can be inserted as needed. Also, multiple stages of geometric processing can be implemented and buffer memories can be inserted therebetween for buffering of processed information.

F

GEOMETRIC PROCESSOR

### General Description

Various geometric processor configurations will be described to illustrate improved geometric processor capability. This improved capability includes hardware efficiency, high speed, and many geometric processing features. Advantages are derived from conceptual features, such as a window having multiple geometric processing operations; from hardware features, such as an improved arrangement of devices; and logical features, such as an improved logical design.

### Coordinate Systems

Image processing will generally be discussed in the form of a rectilinear coordinate system for geometric processing and spatial processing and in the form of a polar coordinate system for topological shadowing and occulting. Other coordinate configurations can be used; such as rectilinear coordinates for all processing, or polar coordinates for all processing, different combinations of rectilinear and polar coordinates for different types of processing, and other alternates thereof.

### Geometric Preprocessing And Postprocessing

Various configurations of image processing are discussed in the form of particular implementations herein. However, the arrangement of the processing elements can be varied to meet the requirements of a particular application. Examples are provided below.

Geometric processing can be implemented as geometric preprocessing for scanning an image into an image memory and as geometric postprocessing for scanning an image out of an image memory. Spatial filtering can be implemented as filter preprocessing for reducing spatial frequencies and as filter postprocessing for smoothing images. Image memories can be provided for buffering images; such as a front end buffer memory, a shading buffer memory, a geometric processing image memory, and a refresh memory. Various devices can be arranged in pipeline fashion for pipeline processing of image information; such as a pipeline of pre-filtering, geometric processing, and post filtering arrangements. Various devices can be arranged in parallel fashion for parallel processing of image information; such as a parallel arrangement of geometric processors for geometrically processing different images to be overlayed together. Various devices can be arranged in parallel pipeline fashion for parallel pipeline processing of image information; such as combining the above parallel processing and pipeline processing together.

Geometric processing is described herein in the configuration of a scanout arrangement, where an image stored in image memory is scanned out and geometrically processed during scanout. For example, the image in image memory may be a normalized image that has not as yet been geometrically processed; where the geometric processor can rotate, translate, expand, compress, generate 3D-perspective, and warp the image during scanout. Also, the geometric processor can pre-process an image; such as to geometrically process a database image for loading a geometrically preprocessed image into image memory. The image can be derived from a database memory, a digitizer, or other source of pixels. The geometric pre-processor can generate memory addresses for loading pixels into image memory in a manner similar to the previously discussed geometric processor that generates memory addresses for scanning pixels out of image memory. This geometric pre-processor implementation has many uses; such as for providing a geometrically processed image in a memory for refreshing a display. For example, an input data stream can be generated from a digitized input; such as from a camera, sonar receiver, or radar receiver; and can be scanned into image memory with geometric preprocessing. The geometrically preprocessed image in image memory can be used to refresh a display, can be used for pattern recognition, and can be used for other purposes. It provides a geometrically preprocessed image that can be processed with other devices without further geometric processing.

The geometric preprocessor may not have to be a real time geometric preprocessor; where it may be implemented to accept relatively low data rate input image information for geometric preprocessing at a relatively low rate for storage in image memory. This can provide various advantages; such as where higher speed processing operations can be used to process the geometrically preprocessed image stored in image memory. For example, a geometrically preprocessed image in image memory can be used to refresh a display without further geometric processing. Also, a digitized image from a sensor can be scanned into image memory as the digitized pixel stream is generated in accordance with geometric processing capability; such as in rotated, expanded, compressed, and warped image form.

Geometric processing has previously been discussed in the form of an address generator that generates the addresses of pixels having a geometrically processed relationship therebetween, such as having a rotated and expanded relationship therebetween. Such address generation is discussed as being accompanied by read operations of the image memory for scanning out the addressed pixels. This address generator can be adapted for geometric preprocessing to scan a geometrically preprocessed image into image memory by generating the geometric processing related addresses accompanied by write operations (in place of the above-mentioned read operations) for writing pixels into image memory in geometrically processed form. Therefore, geometric preprocessing can be implemented with an arrangement similar to that discussed for geometric postprocessing.

Also, geometric postprocessing for scanning out an image from image memory can also be used for scanning a geometrically processed image into a subsequent image memory. The geometric processor can be used to address the previous image memory for scanout and to address the subsequent image memory for scan-in. The implementations can be similar but the capabilities can be different. For example, scan-out from an image memory provides the capability of generating dynamic geometrically processed output images in response to a static image stored in image memory. Scan-in to an image memory provides the capability of generating a static geometrically processed image, such as for refreshing a display. Many other capabilities of scan-in and scan-out geometric processors will become readily apparent from the teachings herein.

For simplicity of discussion, geometric processing discussed herein will be discussed in the context of geometric postprocessing. However, these discussions of geometric postprocessing are also illustrative of geometric preprocessing.

## Window Geometry

One configuration of geometric processing is implemented by computationally defining certain parameters of a window and then scanning out the portions of image memory seen through the window in a raster scan referenced to the window. The window can be translationally positioned over image memory to implement image translation, can be rotationally oriented over image memory to implement image rotation, can be expanded or compressed to implement image compression or expansion respectively, can be warped to implement image warping, and can be otherwise controlled to provide other forms of image processing. Generating a raster scan that is referenced to the window geometry and that accesses the pixels in image memory which correspond to the window geometry generates a geometrically processed image in raster scan form that can be used to directly refresh a CRT monitor without the need for a refresh memory.

Defining of the window can be performed with a supervisory processor deriving certain window parameters on a once per frame or once per field basis. One window parameter is translational position of the window relative to image memory, which can be defined with the image memory coordinate of the corner of the window, the upper left hand corner of the window, at which the raster scan begins. Another window parameter is the rotational position of the window relative to image memory, which can be defined with the ratio of slopes of the X-axis and Y-axis address generators. Another window parameter is the size of the window relative to image memory, which can be defined with the magnitude of the slope parameters. Another window parameter is the change

in the slope parameters, which can be defined with delta slope parameters that update the slope parameters during the raster scan operation. The slope parameters can be implemented to update the address generators and consequently to provide the step sizes as the address generators scan through the window and through image memory. In one implementation, the ratio of the slope parameters defines the rotation of the window and the vector length of the slope parameters defines the size of the window. For example, the square root of the sum of the squares of the component step sizes or slopes defines the vector step size and consequently the window dimensions.

VECTOR STEP = SQR ((X STEP ^2) + (Y STEP ^2))

Similarly, changes in the component step sizes and ratio of the slopes causes warping of the window and consequently warping of the scanned-out image. The "slope" related terminology used herein may be used interchangeably with "delta" terminology relative to Fig 6 herein.

### Image Hierarchy

The image processing system can be configured with a hierarchy of images, a pipeline of images, or other combination of images implemented with a hierarchy of memories, a pipeline of memories, or other combination of memories; one configuration of which will now be described with reference to Fig 2A.

Database memory 204A can be implemented to store a very large image; such as a 1-million pixel image, a 50-million pixel image, a 1/2-billion pixel image, or a 2-billion pixel image. Database memory 204A can store multiple versions of the same image, such as pre-compressed and pre-filtered versions of the same image. Various database memory configurations are discussed herein.

A portion of the image stored in database memory 204A can be loaded into buffer memory 204B. Buffer memory 204B can be used for more rapid access to a portion of the image; for pre-processing, such as for pre-filtering and pre-compression; and for other purposes. The portion of the image loaded into buffer memory 204B can be selected based upon the translated position of window 204D, such as in accordance with the virtual scrolling implementation discussed herein.

A portion of the image stored in the buffer memory 204B in a buffered system or a portion of the image stored in the database memory 204A in an unbuffered system can be loaded into an image memory 204C. Image memory 204C can be used for geometric processing of the image; such as for rotation, translation, expansion, compression, 3D-perspective, and warping of the image. The portion of the image loaded into image memory can be selected

based upon the translated position of window 204D, such as in accordance with the virtual scrolling implementation discussed herein.

A portion of the image stored in image memory 204C can be processed with a window 204D, as discussed below. Window 204D can be rotated, translated, expanded, compressed, 3D-perspective processed, and warped in accordance with geometric processing to provide a correspondingly rotated, translated, expanded, compressed, 3D-perspective, and warped image; such as for display on a monitor.

In one configuration, database memory 204A may have the largest capacity, such as 1-billion pixels, and the longest latency time, such as 1-second, of any of memories 204A to 204C buffer memory 204B may have a smaller capacity, such as 10-million pixels, and a smaller latency time, such as 1-microsecond, then database memory 204A; image memory 204C may have a smaller capacity, such as 1-million pixels, and a smaller latency time, such as 0.1-microsecond, then buffer memory 204B; and window 204D may have a smaller capacity, such as 1/3-million pixels, and a smaller latency time, such as an instantaneous scanout, then image memory 204C.

For simplicity of discussion; the memory hierarchy has been shown in Fig 2A having sizes, orientations, shapes, positions, speeds, capacities, relative characteristics, and other characteristics. However; different sizes, orientations, shapes, positions, speeds, capacities, relative characteristics, and other characteristics can also be provided as an alternate to those described above.

### Description of Fig 2B

A translation and rotation geometric processing arrangement will now be discussed with reference to Fig 2B. Other geometric processing features; such as compression, expansion, and 3D-perspective; can be implemented in conjunction with this rotation and translation configuration. This configuration will be discussed relative to transforming an image stored in an image memory.

In one configuration, the geometric processor discussed herein may be characterized as an address generator that transforms addresses of pixels from a source memory into addresses of pixels for a destination memory to facilitate transferring of pixels from the source memory into the destination memory with a related translation, rotation, and compression characteristic. The destination memory may be implicit in the refreshing of a display monitor and need not be expressly implemented. Address transformation can be implemented in various ways. In one configuration discussed herein, a row address generator and a column address generator generate pixel addresses along a line traversing image memory at an angle and offset within image memory to implement rotation and translation, respectively.

Geometry of window 205A relative to image memory 205B will now be discussed with reference to Fig 2B. Translation and rotation can be implemented with scanning logic that addresses pixels along a scanline having a selectable slope, such as scanlines 205C. This can be achieved with a column address generator by driving the independent variable in incremental

column or pixel steps and by driving the dependent variable in sub-incremental column or pixel steps, where the sub-incremental column or pixel steps are determined by the slope of the line to be generated. The scanning of a plurality of lines offset therebetween 205C can be used to form a multiple scanline image 205D at selectable slope, such as for storing addressed pixels in another memory map or for directly outputting addressed pixels to a raster scan monitor. The plurality of lines 205C can be offset by starting each line at a different start pixel position at the left hand side of the window. This can be achieved with a row address generator by driving the independent variable in incremental row steps and by driving the dependent variable in sub-incremental row steps, where the sub-incremental row steps are determined by the slope of the row line to be generated. A row line 205E can be used to address the start point pixels for the column scanlines.

For simplicity of discussion, the arrangements discussed with reference to Fig 2A establish the independent variable as the X-variable, and the dependent variable as the Y-variable. However, in the BASIC PROGRAM LISTING DIS.ASC provided herein, a configuration is provided that permits either the X-variable or the Y-variable to be defined as the independent variable for the column address generator and for the row address generator.

In the configurations discussed with reference to Fig 2B, for simplicity of illustration, translation is provided by selecting the initial point 205H, which is the pixel position that is used to initialize the row address generator, and

rotation is provided by selecting the slope of the column address generator. For a rectangular viewport, the slope of the row address generator can be selected so that row line 205E is generated to be orthogonal to column lines 205C. The length of column lines 205C may be used to define width of the viewport; i.e., 512-pixels, and the length of row line 205E may be used to define height of the viewport; i.e., 512-lines.

### Description Of Figs 2C to 2F

The geometric processor can be considered to implement a window that is superimposed on image memory, where the window can represent the viewport and the portions of the image seen through the window can be displayed on the viewport. A configuration having such a window arrangement will now be discussed with reference to Figs 2C to 2F.

A window arrangement may be considered as mapping a portion of image memory seen through a window into the display viewport. This is different from windowing of the viewport, where the viewport is divided into a plurality of different areas, called windows, for different images. Translation of the window relative to image memory translates the image seen in the viewport. Expansion or compression of the window relative to image memory expands or compresses, respectively, the image seen in the viewport. Rotation of the window relative to image memory rotates the image seen in the viewport. Warping of the window relative to image memory warps the image seen in the viewport. One configuration of a window is a geometric representation of the portion of image memory encompassed by the scanlines. In this configuration, scanlines commence at the upper left hand corner of the window and progress in raster scan form towards the lower right hand corner of the window. Translation of the window can be defined by the start point of the scanlines relative to image memory. Rotation of the window can be defined by the angle of the scanlines relative to image memory. Size of the window can be defined by the step size along a scanline and the step size between scanlines relative to the pixel coordinates in image

memory. Warping of the window can be defined by the changes in step size, angle of the scanlines, and with other parameters associated with the window. Other parameters can also be used to define the geometric features of the window.

One configuration of window geometry will now be discussed with reference to Figs 2C to 2F. A window ABCD is shown superimposed on image memory. The image in image memory is represented by the page of the figure extending beyond the window outline. The portion of the image contained within the window ABCD is displayed in the viewport. The scanlines can be generated relative to the window. The first scanline begins at the upper left hand corner of the window: point-A; coordinate XIP1, YIP1; and progresses parallel to the top of the window toward point-D. The last scanline begins at the lower left hand corner of the window, point-B, and progresses parallel to the bottom of the window and ends at the lower right hand corner of the window, point-C. Consequently, the non-rotated window scans-out a non-rotated portion of image memory to be displayed in the viewport.

Window ABCD can be rotated about any point; such as at the center of the window; at the upper left hand corner of the window XIP1, YIP1; at an offset from the center of the window that is within the window; or at an offset from the center of the window that is outside of the window. The center point of rotation can be controlled, such as by the supervisory processor. Rotation about the center of the window : point-O; coordinate XC1, YC1: from window orientation ABCD to window orientation EFGH is shown

in Figs 2C and 2E. Rotation about a point that is offset from the center of the window: point-P; coordinate XC2, YC2: from window orientation ABCD to window orientation EFGH will now be discussed with reference to Figs 2D and 2F.

The portion of the image contained within the rotated window EFGH is displayed in the viewport. The scanlines can be generated relative to the rotated window. The first scanline begins at the upper left hand corner of the window: point-E; rotated coordinate XIP1, YIP1: and progresses parallel to the top of the window toward point-H. The last scanline begins at the lower left hand corner of the window, point-F, and progresses parallel to the bottom of the window and ends at the lower right hand corner of the window, point-G. Consequently, the rotated window scans-out a rotated portion of image memory to be displayed in the viewport.

The geometry associated with rotation about the center of the window will now be discussed with reference to Fig 2C. The window has a center point-O; coordinate XC1, YC1; and an initial point for image scanout; coordinate XIP1, YIP1. The dimensions of the viewport are X5V for the X-dimension and Y5V for the Y-dimension. The distance from the scanout start point XIP1, YIP1 to the center of rotation is R1. In this case, R1 is also the diagonal of the window. Hence, the distance from the scanout initial point XIP1, YIP1 to the center of the window is  $R1/2$ . The angle between the diagonal of the window and the horizontal through the center of the window is AP1. The angle AR is the angle of rotation from the angular position of the non-rotated window ABCD to the angular position of the rotated window EFGH.

The distance between the center of the viewport: point-O; coordinate XC1, YC1: and the initial point of the scanout: point-E; coordinate XIP1, YIP1: is delta-X DX and delta-Y DY.

F The geometric relationships shown in Fig 2C are summarized  
in equation form below.

$$\tan(AP1) = AB/BC = Y5V/X5V$$

$$AP1 = \text{ATN}(Y5V/X5V)$$

$$R1 = \text{SQR}(X5V^2 + Y5V^2)$$

$$E0 = EG/2 = AC/2 = R1/2$$

$$DX = (R1/2) * \cos(AP1 - AR)$$

$$DY = (R1/2) * \sin(AP1 - AR)$$

Substituting trigonometric relationships into DX and DY yields:

$$DX = (R1/2) * (\cos(AR) * \cos(AP1) + \sin(AR) * \sin(AP1))$$

$$DY = (R1/2) * (\sin(AR) * \cos(AP1) - \cos(AR) * \sin(AP1))$$

Defining terms yields:

$$KB1 = (R1/2) * \sin(AP1)$$

$$KB2 = (R1/2) * \cos(AP1)$$

Substituting of KB1 and KB2 into DX and DY yields:

$$DX = KB2 * \cos(AR) + KB1 * \sin(AR)$$

$$DY = KB2 * \sin(AR) - KB1 * \cos(AR)$$

Defining terms yields:

$$KF4 = KB1 * \sin(AR)$$

$$KF5 = KB2 * \sin(AR)$$

$$KF6 = KB2 * \cos(AR)$$

$$KF7 = KB1 * \cos(AR)$$

Substituting of KF4, KF5, KF6, and KF7 into DX and DY yields:

$$DX=KF6+KF4$$

$$DY=KF5-KF7$$

Also, XIP1 and XIP2 can be defined as follows:

$$XIP1=XC1-DX$$

$$YIP1=YC1-DY$$

Substituting of the equations for DX and DY yields:

$$XIP1=XC1-KF4-KF6$$

$$YIP1=YC1-KF5+KF7$$

Implementation of this configuration is shown in the BASIC PROGRAM LISTING DIS.ASC provided herein as the old window geometry selected with PR32\$="N".

The geometry associated with rotation about a point that is offset from the center of the window will now be discussed with reference to Fig 2D. This configuration is similar to the geometry discussed above for rotation about the center of the window. The window has a center point-O; coordinate XC1, YC1; a center of rotation point-P; coordinate XC2, YC2; and an initial point for image scanout; coordinate XIP1, YIP1. The dimensions of the viewport are X5V for the X-dimension and Y5V for the Y-dimension. The distance from the scanout start point XIP1, YIP1 to the center of rotation is R1. The angle between the vector R1 and the horizontal through the center of rotation is AP1. The angle AR is the angle of rotation from the angular position of the non-rotated window ABCD to the angular position of the rotated window EFGH. The distance between the center of rotation: point-P; coordinates XC2, YC2: and the start point of the scanout: point-E; coordinates XIP1, YIP1: is delta-X DX and

delta-Y DY. The offset of the center of rotation; coordinates XC2, YC2; from the center of the window; coordinates XC1, YC1; is the distance TX, TY.

The geometric relationships shown in Fig 2D are summarized in equation form below.

$$Q2 = Y5V/2 - TY$$

$$Q3 = X5V/2 - TX$$

$$\tan(AP1) = Q2/Q3$$

$$AP1 = \text{ATN}(Q2/Q3)$$

$$R1 = 2 * \text{SQR}(Q2^2 + Q3^2)$$

$$DX = (R1/2) * \cos(AR - AP1)$$

$$DY = (R1/2) * \sin(AR - AP1)$$

Substituting trigonometric relationships into DX and DY yields:

$$DX = (R1/2) * (\cos(AR) * \cos(AP1) + \sin(AR) * \sin(AP1))$$

$$DY = (R1/2) * (\sin(AR) * \cos(AP1) - \cos(AR) * \sin(AP1))$$

Defining terms yields:

$$KB1 = (R1/2) * \sin(AP1)$$

$$KB2 = (R1/2) * \cos(AP1)$$

Substituting of KB1 and KB2 into DX and DY yields:

$$DX = KB2 * \cos(AR) + KB1 * \sin(AR)$$

$$DY = KB2 * \sin(AR) - KB1 * \cos(AR)$$

Defining terms yields:

$$KF4 = KB1 * \sin(AR)$$

$$KF5 = KB2 * \sin(AR)$$

$$KF6 = KB2 * \cos(AR)$$

$$KF7 = KB1 * \cos(AR)$$

Substituting of KF4, KF5, KF6, and KF7 into DX and DY yields:

$$DX = KF6 + KF4$$

$$DY = KF5 - KF7$$

Also, XIP1 and <sup>YIP1</sup>  
 $\hat{XIP2}$  can be defined as follows:

$$XIP1 = XC1 + TX - DX$$

$$YIP1 = YC1 + TY - DY$$

Substituting of the equations for DX and DY yields:

$$XIP1 = XC1 + TX - KF4 - KF6$$

$$YIP1 = YC1 + TY - KF5 + KF7$$

Implementation of this configuration is shown in the BASIC PROGRAM LISTING DIS.ASC provided herein as the old window geometry selected with PR32\$="N".

Arrangements for rotating about the center of the window and about an offset from the center of the window have been discussed with reference to Figs 2C and 2D respectively, above. An alternate configuration for rotating about the center of the window and about an offset from the center of the window will now be discussed with reference to Figs 2E and 2F, respectively, below. Implementation of this arrangement is shown in the Basic listing DISM922A.ASM transmitted herewith. Particular note should be made of lines 50 to 68 of this listing identifying the changes from the previously referenced listing that implemented the arrangement shown in Figs 2C and 2D.

The geometry associated with rotation about the center of the window will now be discussed with reference to Fig 2E. The window has a center point-0; coordinate XC1, YC1; and an initial point for image scanout; coordinate XIP1, YIP1. The dimensions of the viewport are X5V for the X-dimension and Y5V for the Y-

dimension. The angle AR is the angle of rotation from the angular position of the non-rotated window ABCD to the angular position of the rotated window EFGH. The distance between the center of the viewport: point-O; coordinate XC1, YC1: and the initial point of the scanout: point-E; coordinate XIP1, YIP1: is delta-X DX and delta-Y DY.

The geometric relationships shown in Fig 2E are summarized in equation form below.

$$Q3 = X5V/2$$

$$Q2 = Y5V/2$$

$$DX = Q3 * \cos(AR) + Q2 * \sin(AR)$$

$$DY = Q2 * \cos(AR) - Q3 * \sin(AR)$$

Defining terms for compatibility with the arrangement shown in Fig 2D yields:

$$KB1 = Q2$$

$$KB2 = Q3$$

Substituting of KB1 and KB2 into DX and DY yields:

$$DX = +KB2 * \cos(AR) + KB1 * \sin(AR)$$

$$DY = -KB2 * \cos(AR) + KB1 * \sin(AR)$$

Defining terms yields:

$$KF4 = KB1 * \sin(AR)$$

$$KF5 = KB2 * \sin(AR)$$

$$KF6 = KB2 * \cos(AR)$$

$$KF7 = KB1 * \cos(AR)$$

Substituting of KF4, KF5, KF6, and KF7 into DX and DY yields:

$$DX = KF6 + KF4$$

$$DY = KF7 - KF5$$

*YIP1*  
Also,  $XIP1$  and  $\cancel{XIP2}$  can be defined as follows:

$$XIP1 = XC1 - DX$$

$$YIP1 = YC1 - DY$$

Substituting of the equations for DX and DY yields:

$$XIP1 = XC1 - KF4 - KF6$$

$$YIP1 = YC1 + KF5 - KF7$$

Implementation of this configuration is shown in the BASIC PROGRAM LISTING DIS.ASC provided herein as the new window geometry selected with PR32\$="Y".

The geometry associated with rotation about a point that is offset from the center of the window will now be discussed with reference to Fig 2F. This configuration is similar to the geometry discussed with reference to Fig 2E above for rotation about the center of the window. The window has a center point-O; coordinate XC1, YC1; a center of rotation point-P; coordinate XC2, YC2; and an initial point for image scanout; coordinate XIP1, YIP1. The dimensions of the viewport are X5V for the X-dimension and Y5V for the Y-dimension. The angle AR is the angle of rotation from the angular position of the non-rotated window ABCD to the angular position of the rotated window EFGH. The distance between the center of rotation: point-P; coordinates XC2, YC2; and the start point of the scanout: point-E; coordinates XIP1, YIP1: is delta-X DX and delta-Y DY. The offset of the center of rotation; coordinates XC2, YC2; from the center of the window; coordinates XC1, YC1; is the distance TX, TY.

The geometric relationships shown in Fig 2F are summarized in equation form below.

$$Q2=Y5V/2-TY$$

$$Q3=X5V/2+TX$$

$$DX=Q3*\cos(AR)+Q2*\sin(AR)$$

$$DY=Q2*\cos(AR)-Q3*\sin(AR)$$

F Defining terms for compatibility with the arrangement shown in Fig 2D yields:

$$KB1=Q2$$

$$KB2=Q3$$

Substituting of KB1 and KB2 into DX and DY yields:

$$DX=KB2*\cos(AR)+KB1*\sin(AR)$$

$$DY=KB1*\cos(AR)-KB2*\sin(AR)$$

Defining terms yields:

$$KF4=KB1*\sin(AR)$$

$$KF5=KB2*\sin(AR)$$

$$KF6=KB2*\cos(AR)$$

$$KF7=KB1*\cos(AR)$$

Substituting of KF4, KF5, KF6, and KF7 into DX and DY yields:

$$DX=KF6+KF4$$

$$DY=KF7-KF5$$

Also, XIP1 and YIP1 can be defined as follows:

$$XIP1=XC1+TX-DX$$

$$YIP1=YC1+TY-DY$$

Substituting of the equations for DX and DY yields:

$$XIP1=XC1+TX-KF4-KF6$$

$$YIP1=YC1-TY+KF5-KF7$$

Implementation of this configuration is shown in the BASIC PROGRAM LISTING DIS.ASC provided herein as the new window geometry selected with PR32\$="Y".

Translation can be implemented by redefining the window center point-0; coordinate XC1, YC1. For example, the BASIC PROGRAM DIS.ASC provided herein updates XC1, YC1 as a function of joystick translational commands.

Expansion and compression can be implemented by defining the slope or step size for the window. If the slope parameters are set at unity, the image will be sampled pixel-by-pixel without pixel replication and without undersampling to display a unity size image. If the slope parameters are set to less than unity, the image will be sampled in less than pixel steps; yielding replicated pixels to display an expanded size image. If the slope parameters are set to more than unity, the image will be sampled at greater than pixel steps; yielding undersampling to display a compressed size image.

3D-perspective and warping can be implemented by varying the slope or step size across the window. Setting the step size to vary on a scanline-by-scanline basis from the top of the image to the bottom of the image provides 3D-perspective caused by rotation of the image about the X-axis to tilt the top of the window backward into the third dimension. Setting the step size to vary along a scanline and varying the slope of the scanlines on a scanline-by-scanline basis from the top of the image to the bottom of the image provides 3D-perspective caused by rotation of the image about the Y-axis to tilt the side of the window backward into the third dimension. Other variations of step size,

slope, and other parameters yield other forms of warping. Warping can result in changing a rectangular window to a warped window; such as a trapezoidal window for 3D-perspective.

In view of the above, one method for implementing geometric processing is to define the scanline start point XIP1, YIP1 and the slopes. The ratio of the X-slope and Y-slope, both for row and pixel processing, defines the angle of the scanlines and consequently the rotation of the window. The vector magnitude of the X-slope and Y-slope, both for row and pixel processing, defines the sampling step size of the scanlines and consequently the expansion and compression of the window. The magnitude of XIP1, YIP1 defines the position of the window and consequently the translation of the window.

### Description Of Fig 2G

An alternate window geometry will now be discussed with reference to Fig 2G. This window geometry is an earlier configuration that has been updated with the configurations discussed with reference to Figs 2C to 2F.

The window can be rotated, translated, compressed, and expanded; as implemented in the BASIC PROGRAM LISTING DIS.ASC provided herein; and can be otherwise modified; such as to correct image distortions and to introduce image distortions. Position can be implemented by defining the upper lefthand corner of the window, XIP and YIP, for a particular frame. Translation can be implemented by defining different values for the upper lefthand corner of the window, XIP and YIP, for a sequence of frames. Rotation can be implemented by defining the angle-AR1 and hence implicitly defining the X-slope and Y-slope proportional to DXC and DYC, respectively. Squareness of the window can be implemented by defining the angle-AR2 and hence implicitly defining the X-slope and Y-slope proportional to DXR and DYR respectively. As shown in Fig 2D, if angle-AR1 is equal to angle-AR2, a rectangular window is provided. If angle-AR1 is not equal angle-AR2, a non-rectangular is provided.

The following equations define the window characteristics.

$$DXC = LI * \cos(AR)$$

$$DYC = LI * \sin(AR)$$

$$DXR = -NI * \sin(AR)$$

$$DYR = NI * \cos(AR)$$

$$DX0 = -R1 * \cos(A5) / 2$$

$$DY0 = -R1 * \sin(A5) / 2$$

A1 = ATN(NI/LI)

A5 = AR+A1

These geometric relationships can be readily understood by one skilled in the art. For example, the equations for DXC, DYC, DXR, and DYR are conversions from polar coordinates to rectangular coordinates for the window boundaries. The equations for DX0 and DY0 are conversions from polar coordinates to rectangular coordinates to derive the startpoint coordinates, XIP and YIP, relative to the center point coordinates, XIC and YIC. The equations for the angle-A1 and the angle-A5 represent an arctangent of the window dimensions and an angle summation respectively for deriving the startpoint coordinates from the centerpoint coordinates.

Expansion and compression can be implemented by defining the values of the slope parameters for the address generators. Expansion and compression can also be used to scale the values of DX0 and DY0 to derive the startpoint coordinates from the centerpoint coordinates.

### Description of Figs 2H and 2I

One configuration of translational and rotational geometric processing will now be discussed with reference to Figs 2H and 2I. For simplicity of discussion, Figs 2H and 2I will be discussed relative to dependent and independent variables, Y and X respectively for this example.

The flow diagram and state diagram shown in Fig 2H may be implemented in software and firmware in a stored program computer; may be implemented in hardware such as with the arrangement shown in Fig 2I; or may be implemented in combinations of hardware, firmware, and software. For example, lower processing bandwidth operations can be performed in software, medium processing bandwidth operations can be performed in firmware, and higher bandwidth operations can be performed in hardware.

Operation proceeds to element 210A, where initial point coordinates XR and YR and slopes of the row line MR and the column pixel MC, are initialized. The initial point can be derived in response to translation processing, such as by adding the changes in translational position to the coordinates of the prior initial point to obtain the coordinates of the next initial point. The slopes of column pixels 205C and row lines 205E (Fig 2B) can be derived in response to rotation processing, such as by adding the changes in rotational position to the prior rotational position to obtain the next rotational position. In a system having rectilinear coordinate driving functions, the slopes of row lines 205E and column pixels 205C can be derived by determining which coordinate is the independent variable and then

by dividing the dependent variable by the independent variable. In a system having polar coordinate driving functions, the slope of column pixels 205C can be determined by the tangent of the rotational angle; the slope of row lines 205E can be determined by the cotangent of the rotational angle; and the independent variables and dependent variables can be determined by cosine and sine functions of the rotational angle and line length.

Operation proceeds to element 210B to test for a frame sync pulse, which precedes<sup>c</sup> the start of the first scanline.

Therefore, the frame sync pulse can be used to control loading of initial conditions for next frame <sup>'1</sup><sub>A</sub> between completion of the prior frame and initiation of the next frame. Until the frame sync pulse is detected, operation branches along the NO path to loop around element 210B. When a frame sync pulse is detected, operation branches along the YES path to element 210C to initialize the pixel address generator to the start point of the line. This is achieved by loading the XC, YC, and RC registers for the pixel address generator from the corresponding XR, YR, and RR registers of the row address generator. Therefore, it can be seen that the row address generator generates initial conditions for each scanline. In a hardware configuration, the frame sync pulse can clock the synchronous hardware logic to automatically advance the logic states, where element 210B can be implicit therein.

Operation proceeds to element 210D to test for a line synch pulse, which initiates start of each scanline. Therefore, the line synch pulse can be used to initiate operation of the pixel

address generator. Until the line synch pulse is detected, operation branches along the NO path to loop around element 210D. When a line synch pulse is detected, operation branches along the YES path to element 210E to initiate generation of the next line. In a hardware configuration, the line synch pulse can clock synchronous hardware logic to automatically advance the logic states, where element 210D can be implicit therein.

Operation proceeds to element 210E to test for a clock pulse, which clocks the pixel address generator to step along the scanline for addressing a sequence of pixels in the scanline. Until the clock pulse is detected, operation branches along the NO path to loop around element 210E. When a clock pulse is detected, operation branches along the YES path to element 210F to access the present pixel from image memory and then to elements 210I and 210J to update the pixel address and row address. In a hardware configuration, the clock pulse can clock the synchronous hardware logic to automatically advance the pixel address generator along the line, where element 210E can be implicit therein.

In element 210F, the addressed pixel is accessed from image memory, where the pixel is addressed by the XC and YC coordinates generated by the pixel address counter. The accessed pixel can be loaded into another memory map or alternately can be output directly to a raster scan monitor.

Operation proceeds to element 210G to test for a frame sync pulse. Until a frame sync pulse is detected, operation branches to element 210H to test for a line synch pulse. Until a line synch pulse is detected, operation branches along the NO path to

element 210I for successive generation of pixel addresses. When a line sync pulse is detected, operation branches along the YES path to element 210J to terminate processing of the present scanline and to initiate processing of another scanline. When a frame sync pulse is detected, operation branches to element 210A to terminate processing of the present frame and to initiate processing of another frame.

When a line sync pulse is detected in element 210H, operation branches along the NO path to element 210I to increment the pixel address generator along the scanline to the next pixel in element 210I and to access the next pixel in elements 210E and 210F. The pixel address generator is incremented by incrementing the independent variable, XC in this example, and by updating the dependent variable, YC in this example, by adding the slope-related parameter MC to the dependent variable, YC in this example, to derive the next pixel address, XC and YC.

When a line sync pulse is detected in element 210H, operation branches along the YES path to element 210J to increment the row address generator along the row line to the next scanline start point pixel and to element 210C to initiate processing of the next scanline. The row address generator is incremented by incrementing the independent variable, YR in the example, and by updating the dependent variable, XR in this example, by adding the slope-related parameter MR to the dependent variable, YR in this example, to derive the next row start pixel address, XR and YR.

Processing bandwidth requirements are different for different portions of the arrangement shown in Fig 2H. For example, outer loop frame-related elements 210A and 210B are executed once per frame, such as once per 33,000-microseconds; middle loop line-related elements 210C, 210D, and 210J are executed once per line, such as once per 65-microseconds; and inner loop pixel-related elements 210E to 210I are executed once per pixel, such as once per 0.1-microsecond. Therefore, in this example, outer loop frame-related elements have very low processing bandwidth requirements and can be readily implemented in software, middle loop line-related elements have higher processing bandwidth requirements and can be readily implemented in software or firmware, and inner loop pixel-related elements have much higher processing bandwidth requirements and are preferably implemented in firmware or hardware. Further, in a configuration where outer loop or middle loop elements are implemented in software or firmware, they may be processed concurrently with the processing of inner loop hardware implemented elements. Also, outer loop operations may be performed during the frame retrace period and middle loop operations may be performed during the line synch period during which periods the inner loop pixel-related operations need not be executed. Therefore; outer loop, middle loop, and inner loop operations can timeshare some of the same processing logic.

In view of the foregoing, the discussed image processor configuration will iteratively loop from frame to frame, updating rotational and translational position; will iteratively loop within a frame, generating the start point pixel address for each

row; and will iteratively generate each column pixel address within that row to provide a dynamically rotating and translating image.

A hardware implementation of a translation and rotation configuration will now be discussed with reference to Fig 2I. In this configuration, translation and rotation are performed with row line address generator XR, YR, RR, and MR and by column pixel address generator XC, YC, RC, and MC. Row initial conditions are entered into row registers XR, YR, RR, and MR at the beginning of a frame and are used to generate line initial conditions to be entered into pixel address generator registers XC, YC, RC, and MC at the beginning of each new line. After initial conditions are entered into the pixel address generator, the pixel address generator is controlled to generate addresses of pixels along the scanline at the selected slope for accessing pixels along the scanline from image memory 220I for output either to a refresh memory or directly to a display monitor. At the end of a scanline, the line synchronization pulse increments the row address generator to the next scanline and loads the initial conditions for the next scanline to be generated into the pixel address generator. The pixel address generator then proceeds to generate addresses of pixels along the scanline, as discussed above. In this manner, the row address generator increments from row-to-row for each scanline to generate initial conditions for the pixel address generator and the pixel generator increments from pixel-to-pixel for each clock to generate addresses for the pixels along that scanline.

The row address generator and the pixel address generator can generate row and pixel addresses at selected slopes. The slopes, the row slope MR and the pixel slope MC, are loaded as initial conditions for the particular frame and define the rotation of the displayed image scanned-out of image memory 220I relative to the orientation of the image in image memory. The translational position, defined by the initial XR and YR conditions, are loaded as initial condition for the particular frame and identify the translation of the displayed image scanned-out of image memory relative to the position of the image in image memory. Effectively, these rotation and translation initial conditions place a display window over the image memory map, positioned to the XR and YR initial point and rotated by the MC and MR slopes to provide a translated and rotated portion of the image in image memory for display.

Pixel address generator operation will now be discussed with reference to Fig 2I. The independent variable, shown as the XC coordinate in this example, is incremented for each input clock pulse to progress one increment, such as one pixel, per clock pulse. The dependent variable, shown as the YC-coordinate in this example, is updated with a fractional increment. This is achieved by adding the fractional slope parameter MC to the fractional portion of the YC register, shown as the remainder least significant portion RC of the Y register. As the remainder RC builds up with successive additions of the slope parameter MC, the arithmetic carry from the remainder portion RC of the Y-  
<sup>A</sup>  
register propagates to the YC portion of the Y-register for updating the YC-parameter. In this manner, the X-independent

variable and the Y-dependent variable are updated to generate a line of pixel addresses starting at the initial pixel address for that row and propagating with the selected slope.

Row address generator operation will now be discussed with reference to Fig 2I. The independent variable, shown as the XR coordinate in this example, is incremented for each input line synchronization pulse to progress one increment, such as one row, per synchronization pulse. The dependent variable, shown as the YR coordinate in this example, is updated with a fractional increment. This is achieved by adding the fractional slope parameter MR to the fractional portion of the YR register, shown as the remainder least significant portion RR of the Y register. As the remainder RR builds up with successive additions of the slope parameter MR, arithmetic carry from the remainder portion RR of the Y-register propagates to the YR portion of the Y-register for updating the YR-parameter. In this manner, the X-independent variable and the Y-dependent variable are updated to generate a line of row addresses starting at the initial pixel address for that frame and propagating with the selected slope.

### Image Compression

Image Compression can be implemented with sub-pixel spatial sampling. For example, for an image zoomed to half-size, each other pixel in a scanline can be sampled and each scanline can be spaced 2-pixels apart. In an alternate arrangement, the array is scanned to the desired degree of compression and new pixel intensities are created that are a weighted and mixed combination of adjacent pixels. Weighting and mixing reduces loss of information; reduces erroneous visual effects; and permits fractional, non-integer, compression.

Image compression can be implemented by generating scanlines that are further apart than the pixel spacing and by mixing weighted pixel intensities to interpolate scanline positions between pixel coordinates. Scanline spacing can be generated to subpixel resolution with the incremental-type address generators shown in Fig 2I. Remainder registers can be used to define fractional portions of a pixel position and consequently can be used to determine subpixel weights. Alternately, scanline spacing can be generated to sub-pixel resolution and the subpixels that are traversed can be used to define the weights. Image compression can be implemented as a separate operation, <sup>F</sup>  
<sup>A</sup> separate from other image processing operations. This can be <sup>A</sup> implemented with <sup>A</sup> separate dedicated address generators scanning a memory map to the desired compressed spacing. During scanning, pixel intensities can be weighted and mixed to interpolate between actual pixel intensities to obtain the compressed pixel intensities to sub-pixel resolution.

Alternately, image compression can be combined together with other operations, such as together with rotation and translation operations. This can be implemented with shared address generators scanning a memory map to the desired compressed line spacing. During scanning, pixels derived with translation and rotation address generators can compress the image simultaneously with rotating and translating the image. For simplicity of discussion herein, image compression may be discussed as a separate operation. Also, for simplicity of discussion herein, image compression may be discussed using a horizontal scan; which is also illustrative of image compression implemented with a vertical scan and illustrative of image compression implemented with an angular (non-vertical and non-horizontal) scan.

Compression can be implemented by scanning (sampling) image memory with more than original scanline spacing to subpixel resolution and by weighting and mixing pixel intensities to generate new compressed pixel intensities to compress an image. In one configuration, the image can be progressively compressed, such as compressing the image by 90% for one iteration, then compressing the 90% compressed image by another 90% to 81% for a second iteration, and then compressing the 81% compressed image by another 90% to 72.9% for a third iteration; yielding three 90% compressions for a 72.9% compression. This progressive compression can be performed for many iterations. For such a configuration, it may be desirable to store the compressed images each iteration so that it can be again compressed during subsequent iterations, such as with in-place compression or multiple memory compression. In-place compression stores the

compressed image in the same memory map with the non-compressed image, progressively over-writing or overlaying the non-compressed image. Multiple memory compression stores the compressed image in another memory map separate from the memory map storing the non-compressed image, thereby preserving the non-compressed image.

As discussed above for progressive compression, the same image can be iteratively compressed for multiple iterations. Because it may not be necessary to continuously store and preserve each and every one of the iteratively compressed images, it may be desirable to store a reduced number of stored images; i.e. storing a single compressed image without preserving the non-compressed image or storing a compressed image in combination with preserving a non-compressed image. Storage of a single compressed image without preserving the non-compressed image can be implemented with in-place compression processing. Storage of one or more compressed images and preserving the non-compressed image can be implemented with multiple image memories.

For a configuration having progressive compression in image memory and having overlayed mosaics in image memory; if the image in image memory is compressed and the mosaic being overlayed has not as yet been compressed, there can be a discontinuity in the image. In such a system; a mosaic accessed from the database can be compressed to the level of compression of the image in image memory for overlaying into image memory. Such compression can be implemented in various configurations. In one configuration, the mosaic can be loaded into a buffer memory map and compression

processed for subsequent loading of the compressed mosaic into the image memory. Compression in the buffer memory can be implemented as a single-pass compression or can be implemented as progressive compression.

Because compression involves adjacent pixels, compression of a mosaic without adjacent peripheral pixels can result in a discontinuity around the border of the mosaic. Therefore, a mosaic can be accessed together with a border of additional pixels, where the border pixels are used in compression at the edges of the mosaic. Then, after compression, the border of additional pixels can be removed and the compressed mosaic can be overlayed into image memory. This reduces discontinuities at the mosaic border.

Alternately, the buffer memory map can be eliminated by loading the mosaic into image memory and compressing the mosaic therein. The overlay mosaic can be overlayed in a region of image memory not yet encompassed by the window. Therefore, proper implementation of such overlaying of a non-compressed mosaic onto a compressed image in image memory should not adversely affect the windowed image. Compression processing resources of the image memory can then be used to compress the overlayed mosaic to be consistent with the level of compression of the image in image memory. Compensation for discontinuities at the boundary of the mosaic can be accomplished as discussed above for the buffer memory map arrangement using a border of additional pixels.

### Image Expansion

Image expansion can be implemented with pixel replication. For example, for an image zoomed to twice size, each other pixel can be replicated in succession in a line and each line can be replicated in succession, yielding a grouping of four of the same pixels where a single one of that pixel existed in the non-expanded image. In an alternate arrangement, the array is scanned to the desired degree of expansion and new pixel intensities are created that are a weighted and mixed combination of adjacent pixels. Weighting and mixing reduces aliasing and permits fractional, non-integer, expansion.

Image expansion can be implemented in a manner similar to that discussed for image compression. The image can be scanned to the desired expanded resolution, which is less than one-pixel resolution for expansion in contrast to the greater than one-pixel resolution for compression, and the pixels can be weighted and summed for anti-aliasing, similar to that discussed for spatial compression.

Image expansion can be implemented by generating scanlines that are closer together than the pixel spacing and by mixing weighted pixel intensities to interpolate scanline positions between pixel coordinates. Scanline spacing can be generated to subpixel resolution with the incremental-type address generators shown in Fig 2F. Remainder registers can be used to define fractional portions of a pixel position and consequently can be used to determine subpixel weights. Alternately, scanline spacing can be generated to sub-pixel resolution and the subpixels that are traversed can be used to define the weights.

F  
F  
Image expansion can be implemented as a separate operation,  
separate from other image processing operations. This can be  
implemented with separate dedicated address generators scanning a  
memory map to the desired expanded spacing. During scanning,  
pixel intensities can be weighted and mixed to interpolate  
between actual pixel intensities to obtain the expanded pixel  
intensities to sub-pixel resolution.

Image expansion can, alternately, be combined together with  
other operations, such as together with rotation and translation  
operations. This can be implemented with shared address  
generators scanning a memory map to the desired expanded line  
spacing. During scanning, pixels derived with translation and  
rotation address generators expand the image simultaneously with  
rotating and translating the image. For simplicity of discussion  
herein, image expansion may be discussed as a separate operation.  
Also, for simplicity of discussion herein, image expansion may be  
discussed using a horizontal scan; which is also illustrative of  
image expansion implemented with a vertical scan and illustrative  
of image expansion implemented with an angular (non-vertical and  
non-horizontal) scan.

Expansion can be implemented by scanning (sampling) image  
memory with less than original scanline spacing to subpixel  
resolution and by weighting and mixing pixel intensities to  
generate new expanded pixel intensities to expand an image. In  
one configuration, the image can be progressively expanded, such  
as expanding the image by 110% for one iteration, then expanding  
the 110% expanded image by another 110% to 121% for a second

iteration, and then expanding the 121% expanded image by another 110% to 133% for a third iteration; yielding three 110% expansions for a 133% expansion. This progressive expansion can be performed for many iterations. For such a configuration, it may be desirable to store the expanded images each iteration so that it can be again expanded during subsequent iterations, such as with in-place expansion or multiple memory expansion. In-place expansion stores the expanded image in the same memory map with the non-expanded image, progressively over-writing or overlaying the non-expanded image. Multiple memory expansion stores the expanded image in another memory map separate from the memory map storing the non-expanded image, thereby preserving the non-expanded image.

As discussed above for progressive expansion, the same image can be iteratively expanded for multiple iterations. Because it may not be necessary to continuously store and preserve each and every one of the iteratively expanded images, it may be desirable to store a reduced number of stored images; i.e. storing a single expanded image without preserving the non-expanded image or storing an expanded image in combination with preserving a non-expanded image. Storage of a single expanded image without preserving the non-expanded image can be implemented with in-place expansion processing. Storage of one or more expanded images and preserving the non-expanded image can be implemented with multiple image memories.

For a configuration having progressive expansion in image memory and having overlayed mosaics in image memory; if the image in image memory is expanded and the mosaic being overlayed has

not as yet been expanded, there can be a discontinuity in the image. In such a system; a mosaic accessed from the database can be expanded to the level of expansion of the image in image memory for overlaying into image memory. Such expansion can be implemented in various configurations. In one configuration, the mosaic can be loaded into a buffer memory map and expansion processed for subsequent loading of the expanded mosaic into the image memory. Expansion in the buffer memory can be implemented as a single-pass compression or can be implemented as progressive compression.

Because expansion involves adjacent pixels, expansion of a mosaic without adjacent peripheral pixels can result in a discontinuity around the border of the mosaic. Therefore, a mosaic can be accessed together with a border of additional pixels, where the border pixels are used in expansion at the edges of the mosaic. Then, after expansion, the border of additional pixels can be removed and the expanded mosaic can be overlayed into image memory. This reduces discontinuities at the mosaic border.

Alternately, the buffer memory map can be eliminated by loading the mosaic into image memory and expanding the mosaic therein. The overlay mosaic can be overlayed in a region of image memory not yet encompassed by the window. Therefore, proper implementation of such overlaying of a non-expanded mosaic onto an expanded image in image memory should not adversely affect the windowed image. Expansion processing resources of the image memory can then be used to expand the

overlaid mosaic to be consistent with the level of expansion of the image in image memory. Compensation for discontinuities at the boundary of the mosaic can be accomplished as discussed above for the buffer memory map arrangement using a border of additional pixels.

Expanding of large images to view portions of the images to greater resolution can be achieved with the image expansion capability previously discussed. For example, database information can be accessed for loading into image memory and can be expanded to fit into the appropriate resolution in image memory, then can be processed for expansion using the image processor under operator control. The pixel resolution in the database can be matched with the pixel resolution in the image memory to store one database pixel for each image memory pixel. This provides the highest spatial frequency available to the system, as limited by the database resolution. The image can be expanded further than the resolution in image memory with pixel replication, which provides larger images but may not increase the spatial frequencies nor the smoothness of the image. The image can be expanded further than the resolution in image memory with spatial filtering to expand the image beyond the pixel resolution in image memory and to anti-alias or smooth the image so that the pixels are not replicated for expansion but are weighted and smoothed. Interpolation can be used to provide such smoothing.

## Large Image Expansion And Compression Processing

Examining of a large image may involve zooming out to see a large portion of the image with lower detail, panning to a feature of interest, rotating to reorient the feature, and then zooming in on the feature to see it in more detail. In many cases, it will be desirable to zoom out to show the full image on the display; which, for large database images, can involve 1-billion pixels compressed to 1/3-million pixels, instantaneously. Undersampling may not provide the desired capability because it loses too much detail and it introduces undesirable aliasing effects. Conventional disk latency may not provide the desired capability because it can involve excessive amounts of time to access large numbers of pixels for compression, such as requiring about 16-minutes to access 1-billion pixels for compression based upon a 1-megabyte disk access rate. Image compression with anti-aliasing could further increase the time required to compress such a large image.

The system of the present invention can operate in real time, even for the extremes of compressing any selected amount from 1-billion pixels to a single pixel or expanding any selected amount from any one of 1-billion pixels being expanded to fill the viewport. This feature accommodates extremes of expansion and compression, even for ranges of a billion to one, with anti-aliasing in real time interactively with an operator.

Compression processing will now be discussed. Compression of large images to view extended portions of the images can be achieved with the image compression capability previously discussed. For example, database information can be accessed for

loading into image memory and can be compressed to fit into the appropriate resolution in image memory, then can be processed for compression using the image processor under operator control.

One arrangement for compression anti-aliasing previously discussed uses a filtering kernel of pixels and weights. For compression within the dimensions of the kernel, anti-aliasing will be most effective. For compression near the extremes of the kernel or beyond the extremes of the kernel, undersampling effects occur. Overlapping of kernels and shading of the kernel window may be desired to reduce aliasing. For example, anti-aliasing may be implemented with a 3-pixel-by-3-pixel kernel overlapping other kernels by 1-pixel on each side and shaded so that the pixels have lower weights as they get further from the center pixel.

The kernel range for anti-aliasing may vary as a function of the shading across the kernel. For an unshaded kernel, the full kernel size may be the range; i.e., an unshaded 3-pixel by 3-pixel kernel may have a 3-pixel range. For a shaded kernel, such as a cosine shaded kernel, part of the kernel size may be the range; i.e., a shaded 3-pixel by 3-pixel kernel may have a 2-pixel range.

For compressions beyond the limits of a kernel, such as compression beyond a factor of 2 for a 9-pixel kernel; anti-aliasing can be applied by progressive or iterative compression. For example, an image can be compressed to half size in the first iteration with proper anti-aliasing, then can be compressed by another factor of 2 to 1/4th in a second iteration, then can be

compressed by another factor of 2 to 1/8th size in a third iteration, and so forth for progressive <sup>compression</sup> progression during successive iterations. Each iteration performs anti-aliasing in an effective manner because the compression for each iteration is within the anti-aliasing capabilities of the kernel. In one configuration, the number of iterations to obtain a particular degree of compression with a particular kernel size can be calculated from the following equation

$$K=B \text{ exponent } N$$

where K is the size reduction, i.e. K=2 for a factor of 2 size reduction; B is the kernel range for overcoming anti-aliasing, i.e. B=2 for a 3-pixel kernel; and N is the number of compression iterations, i.e. N=4 for 4-compression iterations. For example, for a 3-pixel kernel having an effective compression range of 2-times for each iteration and for 4-iterations, a compression with anti-aliasing of 2 exponent 4 or 16-times can be achieved with anti-aliasing. Similarly, a 25-pixel kernel may be able to provide 3-times compression with anti-aliasing per iteration; permitting a 3-fold reduction in size for one iteration, a 9-fold reduction in size for 2-iterations, and a 27-fold reduction in size for 3-iterations.

As pixels are progressively compressed, the ability of the compression processor increases exponentially with a square-law type effect. For example, a frame compressed to half-size is compressed to 1/4 of the number of pixels. Therefore, a compression processor having a certain pixel processing rate can compress an uncompressed image to 1/2-size in 1-unit of time, can compress the 1/2-size image to 1/4-size in 1/4-unit of time, and

can compress the 1/4-size image to 1/8 size in 1/16th unit of time.

In view of the above, compression processing bandwidth is dominated by the first iteration and compression processing bandwidth for subsequent iterations becomes exponentially lower, as shown in the COMPRESSION PARAMETERS TABLE For example, for a 9-pixel kernel providing iterative compressions to half-size for a 1000-frame image, as shown in the left hand columns of the COMPRESSION PARAMETERS TABLE; the first iteration of half-sized compression takes 1000-iterations for reduction of the 1000-frames to half-size while all the other reductions down to 1/32nd size takes only about 332 iterations ( $250+62.5+15.625+3.9063$ ).

Also, for a 16-pixel kernel providing iterative compressions to third-size for a 1000-frame image, as shown in the middle columns of the COMPRESSION PARAMETERS TABLE; the first iteration of third-sized compression takes 1000-iterations for reduction of the 1000-frames to third-size while all the other reductions down to about 1/81th size takes only about 125-iterations ( $111+12.3+1.4+0.15$ ).

Also, for a 25-pixel kernel providing iterative compressions to quarter-size for a 1000-frame image, as shown in the right hand columns of the COMPRESSION PARAMETERS TABLE; the first iteration of quarter-sized compression takes 1000-iterations for reduction of the 1000-frames to quarter-size while all the other reductions down to 1/64th size takes only about 66-iterations ( $62.5+3.9+.2$ ).

In view of the above, compression processing bandwidth is dominated by the first compression iterations and reduces exponentially as the progressive compression proceeds. This characteristic facilitates enhancing of the compression processing bandwidth and the time required for compression of large images to small size; examples of which will be provided below.

Processing bandwidth requirements can be reduced by storing progressively compressed images, such as in the database. The stored images can be compressed on an off-line basis when preparing the database and can be accessed in real time during on-line operations for interactive real time operation. For example, the COMPRESSION PARAMETERS TABLE shows compression for a 9-pixel kernel compressing 1000-frames, involving 1332-compression iterations to obtain compression to 1/32nd size. However, if a half-size image was pre-generated and stored, the half-size image could be accessed as a starting point for subsequent progressive progressions, reducing the number of iterations from 1332-iterations to 332-iterations, a reduction to about 25% of the previous processing bandwidth and processing time. Additional memory requirements for storing more highly compressed images are relatively small, increasing from the 1000-frame full size image storage requirement to a 1250-frame full size plus half-size image storage requirement. Therefore, a 25% increase in storage capacity can provide a 4-fold increase in processing speed or a 4-fold reduction in processing bandwidth for higher compressions. Similarly, further progressively compressed images can be stored for still further reductions in

processing bandwidth requirements and compression time. For example, storage of 1000 uncompressed frames, 250-compressed frames of half-size, 63-compressed frames of 1/4-size, 16-compressed frames of 1/8-size, and 4-compressed frames of 1/16-size permits obtaining of any compression level down to 1/32nd size in binary steps virtually instantaneously, limited by database latency time for a single frame of a small group of frames.

Compression levels <sup>in</sup><sub>between</sub> these binary compression steps can then be obtained with the geometric processor from the image in image memory. This is because the image having the appropriate binary compression level can be loaded into image memory from database memory and this loaded image can then be compressed to any level down to the next binary step size substantially instantaneously on-the-fly.

Storage of multiple progressively compressed images, in the above example, requires only about a 33% increase in storage capacity for binary compression, yet can provide a significant enhancement in compression processing bandwidth requirements and compression time. Using this arrangement for storing multiple compressed images, rapid compression to any level can be obtained virtually instantaneously. It is achieved by accessing of frames from the database memory having the proper pre-compressed image, loading the pre-compressed accessed frames into image memory, and then compressing to any level <sup>in</sup><sub>between</sub> the pre-compressed steps with the geometric processor. Typically, the access time of the disk for an image the size of the viewport or a reasonable number

of multiples thereof can be performed in less than a second. Compression from the pre-compressed image stored in disk memory to an <sup>in</sup><sub>between</sub> level can be performed within the next frame time using the geometric processor.

Even with this significant improvement in processing bandwidth requirements and processing time requirements, the increase in database memory storage capacity is small; converging on a exponential asymptote-type of function which may require only a fractional portion of the memory requirements for the uncompressed image to store pre-compressed images. The TOTALS for the START column in the COMPRESSION PARAMETERS TABLE identifies the total amount of disk memory needed to store the frames for all compression steps for a 1000:1 compression dynamic range. For 9-pixel kernel, 16-pixel kernel, and 25-pixel kernel configurations; this total is only about 33%, 12%, and 6% more than the storage capacity needed for the 1,000-frame uncompressed image.

COMPRESSION PARAMETERS TABLE

KERNEL <u>SAMPLING</u>	9-PIXELS		16-PIXELS		25-PIXELS	
	1/2	1/3	1/4	1/4	1/4	1/4
<u>ITERATION</u>	<u>FRAMES</u>		<u>FRAMES</u>		<u>FRAMES</u>	
	<u>START</u>	<u>FINISH</u>	<u>START</u>	<u>FINISH</u>	<u>START</u>	<u>FINISH</u>
1	1000	250	1000	111.111	1000	62.5
2	250	62.5	111.111	12.3457	62.5	3.9063
3	62.5	15.625	12.3457	1.3717	3.9063	0.2441
4	15.625	3.9063	1.3717	0.1524		
5	3.9063	0.9766			1066.4063	65.8404
<u>TOTALS</u>	1332.0313	333.0079	1124.8284	124.9808		

The spatial compression arrangement using stored pre-compressed images, as previously discussed, has profound advantages. For example, if a large image, such as the 1000-frame image provided in the previous example, were to be accessed from image memory and compressed for viewing by an observer; processing and compression time would be relatively long and may be excessive for real time operations. For example, accessing of 1000-frames of information for image memory could take an excessive amount of time. This is because disk memories have data rates in the region of megabits per second and a large image may have nearly a billion bits; relating to hundreds of seconds, possibly 10-minutes, to access 1000-frames of image information. Assuming that compression can proceed concurrently with accessing from a disk memory and hence not add to the compression delay; minutes of time are clearly excessive in a system configured to operate in real time.

The arrangement discussed herein using storage of pre-filtered and pre-compressed images can compress from a very large image; i.e., 1000-frames; to a single frame virtually instantaneously; such as within the latency time associated with random access of a single frame from disk memory, which may be less than 1-second of time. In this example, accessing of pre-compressed pre-filtered image information can provide significant advantages, such as 1000-fold reduction in time required to compress and filter a large image to a single frame. Storage of pre-filtered and pre-compressed images, as discussed above, can provide real time operation; even for compression from a very large high resolution image to a very small highly compressed low

resolution image. Even with such a significant compression speed enhancement, the implementation may be simple. For example, a pre-compression processor can be used to pre-filter and pre-compress the image for storage in the database. The processor can be relatively slow in speed compared to an on-line filter and compression processor because pre-filtering and pre-compression can be performed on a background task basis; such as when the image is being loaded into the database or when the task is being set up at the workstation. Also, the amount of extra memory needed to store the pre-filtered and pre-compressed images may be only a fraction of the memory needed to store the uncompressed image; as discussed with reference to the COMPRESSION PARAMETERS TABLE.

## Composite Geometric Processing

The geometric processing features discussed separately herein; in particular translation, rotation, compression, and expansion; will now be discussed for subpixel resolution in a composite configuration.

Translation can be implemented to subpixel resolution by defining the initial point to subpixel resolution.

Rotation can be implemented to subpixel resolution by generating the independent variable to unity pixel resolution and by generating the dependent variable to subpixel resolution, where the ratio of the dependent variable slope (such as less than unity) to the independent variable slope (such as unity) coordinates to subpixel resolution provides an image rotated at an angle. Implicit in rotation, many of the new pixel coordinates fall <sup>"</sup>inbetween prior pixel coordinates, where such inbetween coordinates are provided as the subpixel bits in the rotated X and Y coordinates.

Compression can be implemented to subpixel resolution by setting the slope of the independent variable to be greater than unity and by setting the slope of the dependent variable so that the ratio of the slope of the dependent variable to the slope of the independent variable is equal to the slope of the image.

Expansion can be implemented to subpixel resolution by setting the slope of the independent variable to be less than unity and by setting the slope of the dependent variable so that the slope of the dependent variable to the slope of the independent variable is equal to the slope of the image.

Therefore, the slope of the independent variable can define the amount of expansion or compression to subpixel resolution, the slope of the dependent variable (relative to the slope of the independent variable) can define the amount of rotation to subpixel resolution, and the initial point coordinates can define the amount of translation to subpixel resolution.

An image starts at the initial point to subpixel resolution and progresses with an expansion or compression defined by the independent variable and rotation defined by the dependent variable; traversing pixels to subpixel resolution. Traversing of a pixel provides a pixel coordinate to subpixel resolution, where the integer portion of the X and Y coordinates define the pixel and where the fractional portion of the X and Y coordinates define the offset from the center of the pixel and hence define the subpixel resolution and the anti-aliasing weights. Because the fractional portion is a function of the translation, rotation, and expansion or compression; anti-aliasing relative to this fractional portion provides composite anti-aliasing for rotation, translation, and expansion or compression.

A pixel traversed by a scanline may be non-selected (jumped over) by the address generator, such as with compression processing having sampling step size greater than unity. The effect of such a non-selected pixel on the image is not lost, such as would result from undersampling; where anti-aliasing can combine the effects of such pixels (weighted and summed) into selected pixels. Undersampling can result for compression if the amount of compression exceeds the number of pixels contained in the anti-aliasing kernel. For example, if the anti-aliasing kernel

is a three-by-three kernel, compression can be provided without substantial undersampling up to a compression factor of 2-times or 3-times, but will start introducing undersampling effects if the compression factor exceeds 2-times or 3-times. However, large compression factors can be implemented with progressive compression discussed herein because progressive compression implements compression by a relatively small compression factor that is performed iteratively to obtain a relatively large compression factor.

The arrangement shown in Fig 2I has been discussed for rotation and translation processing without expansion or compression processing. Expansion and compression processing can be provided by adding a slope register and a summer in conjunction with the independent variable registers in the same manner that a slope register and a summer are provided with the dependent variable register. In this manner, the independent variable can be controlled to progress with a step size other than unity, such as greater than unity for compression and less than unity for expansion. The circuit implementation for the dependent variable need not be changed, but the ratio of the dependent variable to the independent variable should be set to be the slope of the rotation desired; i.e. to the tangent of the rotation angle.

Expansion and compression will now be discussed with reference to Figs 2J and 2K. Figs 2J and 2K show integer pixel positions in the Y-direction with solid lines spaced apart in the vertical direction and fractional pixel positions in the Y-

direction with dashed lines spaced apart in the vertical direction.

Fig 2J shows a Y-independent variable progressing in steps less than unity with dashed lines, 0.75-pixel steps, for expansion. The first step progresses from an 0.00-fractional position to an 0.75-fractional position in the first pixel, the second step progresses from an 0.75-fractional position in the first pixel to a 0.50-fractional position in the second pixel, the third steps progresses from an 0.50-fractional position in the second pixel to an 0.25-fractional position in the third pixel, and the fourth step progresses from an 0.25-fractional position in the third pixel to an 0.00-fractional position. This involves 4-steps of the independent variable to traverse 3-pixels, thereby implementing expansion. The fractional position within the pixels can be used for anti-aliasing processing to provide smooth continuous expansion.

Fig 2K shows a Y-independent variable progressing in steps greater than unity with dash lines, 1.25-pixel steps, for compression. The first step progresses from an 0.00-fractional position to an 0.25-fractional position in the second pixel, the second step progresses from an 0.25-fractional position in the second pixel to an 0.50-fractional position in the third pixel, the third steps progresses from an 0.50-fractional position in the third pixel to an 0.75-fractional position in the fourth pixel, and the fourth step progresses from an 0.75-fractional position in the fourth pixel to an 0.00-fractional position. This involves 3-steps of the independent variable to traverse 5-pixels, thereby implementing compression. The fractional position

within the pixels can be used for anti-aliasing processing to provide smooth compression.

### Address Generator Scaling

An example of address generator scaling will now be provided with reference to the configuration shown in Fig 2L. For convenience of discussion, this configuration is shown with 16-bit registers for the X-register and the Y-register. Also, this configuration is shown having 9-bits of integer resolution and 7-bits of fractional resolution.

Fig 2L illustrates the scaling for the X-register, the Y-register, and one slope register. The integer portion of the X-register and Y-register provides 9-bits of pixel resolution for addressing 1-pixel out of 512-pixels in the X-direction and 1-row out of 512-rows in the Y- direction, respectively. This provides for addressing all of the pixel words in a 512-pixel by 512-pixel memory map with the integer portions of the X-register and the Y-register. The fractional portions of the X-register and the Y-register relate to subpixel resolution. The most significant fractional bit represents half-pixel resolution, the second most significant fractional bit represents quarter-pixel resolution, the third most significant fractional bit represents eighth-pixel resolution, and so forth.

A representative slope register is shown with a 2-bit integer portion and a 7-bit fractional portion. The 2-bit integer portion of the slope registers line-up with the least significant 2-bits of the integer portion of the X-register and the Y-register and the 7-bit fractional portion of the slope registers line-up with the 7-bit fractional portion of the X-register and Y-register. Therefore, for this example; as the slope register parameters are added to the X-register and Y-

register, the maximum update per iteration is almost 4-pixels for almost 4-times compression and the minimum update per iteration is 2 exponent-7 for very large expansion.

In view of the above, the parameters in the slope registers are added to the parameters in the X-register and Y-register to address the next pixel.

The independent and dependent variable arrangement discussed with reference to Fig 2I will now be addressed. For the independent variable; the larger the number in the slope register, the more compressed will be the image, and the smaller the number in the slope register, the more expanded will be the image. For the independent variable, a unity integer in the slope register provides unity size transformation, a number greater than unity in the slope registers provides compression, and a number less than unity in the slope register provides expansion. For the dependent variable, the larger the number in the slope register, the greater the rotation angle and the smaller the number in the slope register, the smaller the rotation angle. For the dependent variable, a zero in the slope register (an all zero fractional portion and an all zero integer portion) provides a zero rotational angle; where the scanline is parallel to the coordinate axis of the memory map.

The address generators, discussed with reference to Fig 2I and Fig 2L, will now be discussed for the detailed logical implementation shown in Fig 6O to 6R. Four address generators are shown implemented in Fig 2I; the X-row address generator, the Y-row address generator, the X-pixel address generator, and the Y-

pixel address generator. These four address generators can be implemented to be substantially the same. Therefore, the configuration shown in Fig 60 to 66R shows 4-address generators to implement the X-row address generator, the X-pixel address generator, the Y-row address generator, and the Y-pixel address generator. In an alternate configuration, the X-row address generator and Y-row address generator can be implemented under program control in the supervisory processor. However, for simplicity of experimentation, the X-row address generator and the Y-row address generator are shown implemented in hardware.

The address generators in the geometric processor are implemented to provide various geometric capabilities; such as translation, rotation, expansion, compression, and 3D-perspective. One configuration uses address generators for X and Y address parameters having a slope parameter with good sub-pixel resolution. This facilitates high resolution rotation, expansion, and compression capability. Alternate configurations may be used. For example, counters may be used for the address generator to update the address from pixel-to-pixel as the image is scanned-out. Many counter configurations do not have the flexibility of the present adder arrangement, but are adequate for some applications.

A hardware configuration is provided herein for generating addresses to access image memory and to perform post processing on the information accessed from image memory for real time operation. Alternately, softwired arrangements can be provided for other applications, such as for non-real time applications and such as for applications having lower cost requirements. Such

applications, can be implemented under software control, such as implemented in the software emulator previously described.

### Image Warping

Warping of an image can be useful in various applications. For example, images can be compared by identifying corresponding elements and by warping the image to cause the corresponding elements to match-up or register with the corresponding elements of another image. Images that are distorted, such as with lens distortions, can be corrected by warping to an undistorted form; warping can be used to compensate for characteristics of equipment, such as compensating for perturbations in a video scan by warping the image in the reciprocal direction. Warping can be used to provide 3D-perspective.

Image warping can be implemented with the address generators previously discussed, modified for higher-order function generation. For example, the X and Y pixel address generators have been discussed for providing a first-order curve along a scanline through image memory. Alternately, each scanline can be controlled to follow a higher-order curve through image memory by adapting the address generators to trace out higher order curves. For example, the address generators can generate second-order, third-order, and still higher-order curves and can even generate curves that are sinusoidal, arc-sinusoidal, tangential, arc-tangential, exponential, algorithmic, and other mathematical and non-mathematical functions.

The address generators previously discussed can be considered to have multiple orders of operation. The zero-order can be considered to be a defined point, stored in the X and Y pixel address registers as initial conditions. The first-order can be considered to be the slopes that update the defined point;

where a zero-order address is updated by a first-order or linear coefficient representing the change in position, which is stored in the first-order change (slope) register. The second-order can be considered to be a change in the first-order parameter and can be stored in a second-order change register or second-order delta register. Higher-order corrections can be implemented by coefficients stored in higher-order change registers, where a higher order change register can provide the change per iteration in the next lower order change register.

In an alternate configuration, an equation can be implemented; such as the equation

$$A + Bt + Ct^2 + Dt^3 \dots$$

having a constant term, a first-order term, a second-order term, a third-order term, and higher-order terms with related coefficients. This equation can be implemented in various forms, such as in an incremental implementation. The solution of this equation can be generated as the sum of the products of the coefficient and the various order terms.

Implementing the configuration shown in the Figs 2H, 2I, and 6O to 6R with a second-order term for updating the first-order slope term can provide a parabola-type second-order curve. If the X and Y slope parameters are incremented with feedback, a <sup>circular</sup> <sub>elliptical</sub> type-curve can be implemented.

One application of warping is to provide the effect of a textured image on a curved surface. This can be provided with a background color having a rectangular image placed thereon that is then warped with a higher-order warping geometric processor.

The warped image can be translated in X and Y, can be rotated, and can be otherwise processed to provide a 2D image wrapped on a 3D surface.

In the experimental system, the address generators on logic board BL1 (Figs 6O to 6R) can be upgraded to provide warping capability. This can be performed with the addition of delta-slope registers and adders for iteratively adding delta-slope parameters into the slope parameters to change the slope parameters. The delta-slope parameters can be a constant for the particular problem and need not be updated every frame. However, the slope registers may need to be updated each frame, either by outputting from the computer or by loading from a slope buffer register that is pre-loaded by the computer. Also, the pixel slope registers may have slope buffer registers that are loaded once per frame by the supervisory processor for reloading the pixel slope registers each line.

The delta-slope registers can be characterized as a second-order slope register; which can be updated to higher orders, such as a third-order and a fourth-order. The delta-slope registers, if they store constant parameters for a second-order correction configuration, need not be updated but may be a constant. Alternately, in a third-order correction configuration, the second-order delta-slope parameters can be updated by third-order delta-delta-slope parameters; where the delta-slope registers are updated from a delta-delta slope registers.

In view of the above, the warping capability can be performed by adding delta-slope registers for a second-order correction and a buffer register for storing the slope

parameters. The slope parameter can be reloaded for each scanline, such as in the manner that the pixel position registers are loaded each scanline. This can be performed for the pixel address generators, or the row address generators, or both the pixel and row address generators.

The higher order slope configuration can be placed on a single custom IC chip with the geometric processor because they do not require additional pinouts and because they require only a small amount of additional circuitry. Pinouts can be limited because the higher-order warping registers can be loaded over a shared computer bus and the processing can be performed in conjunction with the address generators that are on the same chip.

The experimental system can be easily expanded to add additional slope resolution. This additional resolution can be 16-bits, 15-bits plus sign, to take advantage of the integer word length in the Basic compiler. As discussed herein with reference to Fig 2L; the slope registers are implemented with 8-bits of subpixel resolution and 3-bits of pixel resolution plus sign. This leaves 4-bits available in a Basic integer number (2, 6-bit bytes are presently used). Slope register loading can be increased from 6-bits per byte to 8-bits per byte without additional computer processing bandwidth because the computer is processing a 16-bit integer word and is outputting 2, 8-bit bytes for the configuration shown in Figs 60 to 6R.

Warping can provide a circular scan pattern. The image to be displayed can be placed in image memory and can be scanned-out with a circle generator for X-pixel and Y-pixel addresses. Row addresses can be linear addresses, representing a radial line through a circular image. Alternately, row addresses can be implemented in circular form or other form to give the appropriate scanline startpoints. For a linear radial row generator, the row generator can generate a startpoint address along a radial line through the circular scan. Pixel address generators can be interconnected therebetween as a circle generator to generate sine and cosine parametric equations of a circle; with the slope registers being updated from the overflow of the <sup>orthogonal</sup><sub>orthonormal</sub> pixel address registers. A circular address generator can start with the innermost circle at the center of the scan and generate circular scans radially outward. A blanking signal can be used to blank the image for circular scans outside of the outermost concentric scanline for progressive radial propagation.

In the above, the circle address generator can be used to map a rectilinear image into a circular image in a refresh memory for refreshing the display.

## Foreground And Background Tasks

The geometric processor can perform dynamic processing as both, foreground tasks and background tasks. Signals needed to refresh a display can be generated as foreground tasks in real time. Signals not needed to refresh a display can be generated as background tasks in near real-time. Foreground tasks can include translation, rotation, and anti-aliasing. Background tasks can include image compression, expansion, shadowing, occulting, data decompression, and graphics generation. As the geometric processor is performing foreground tasks to refresh a display, background tasks can be executed on a time available basis. For example, while the geometric processor is translating, rotating, and anti-aliasing pixels from image memory in real time at the pixel rate, <sup>in</sup>between processing of pixels in real time; the geometric processor can perform compression/expansion processing on the information in image memory by scanning through image memory in non-real time on a time available basis and compressing the image therein.

Time may be available during line sync periods and during frame sync periods for background processing. Also, time may be available during foreground processing during pixel processing periods, permitting concurrent background processing. For example, time may be available to compress 100-pixels in image memory during each horizontal and each vertical sync period and time may be available to compress 1-pixel in image memory during each real time pixel period. Assuming that image memory is 4-times as large as the window, such background processing would need only a few frame periods to compress a new image into image

memory. If compression is a gradually progressing operation, any small time domain discontinuities may not be apparent to the observer. For example, if compression is progressing at a twentieth of a line per frame period, 4-frame periods would involve a discontinuity of about a fifth of a line of compression. Compression of a fifth of a line each four frame periods, about an eighth of a second, would appear as continuous motion.

### Virtual Scrolling And Wrap-Around

Virtual scrolling can provide continuous "seamless" motion over a very large image. For example, a billion pixels can be stored in the database in mosaic form. Mosaics can be accessed from the database 204A as needed in accordance with geometric processing and overlayed in image memory 204C, such as to permit translation through the image up to and beyond the edges of image memory 204C (Fig 2A). Image memory wrap-around provides continuity past the edges of image memory for "seamless" scrolling. Portions of the database image can be overlayed into image memory for geometric processing. In some configurations, overlaying need only be performed at a low rate consistent with translational rate, thereby simplifying the database memory interface and facilitating continuous motion with unlimited branching capability.

*F* <sup>The</sup> ~~As~~ the image memory provides a fast random access store for image processing. The imagery for the total environment can be stored as mosaics in the database. An active portion of the database can be resident in image memory for geometric processing. As motion of the display window causes the image being displayed to approach an edge of image memory, new mosaics can be accessed from the database and overlayed into available portions of image memory. As window motion passes the edge of image memory, the window can wrap around to the other side of image memory containing the continuing portion of the image for "seamless" image continuity. This will be termed "virtual scrolling".

The database can be stored in a database memory; such as a <sup>W</sup>winchester disk, video disk, or video tape. A video disk or tape can store over 1-billion pixels; i.e. about 4,000 medium resolution frames. A <sup>W</sup>winchester disk can store up to 1/4-billion pixels; i.e. about 1,000 medium resolution frames. In many applications, the database access rate requirements are low; such as being determined by the time to translate an image across image memory. This permits low access rate memories, such as a sequential access video tape, to be used for image storage.

Operation of a window moving relative to an image memory has been previously discussed, such as relative to Figs 2A to 2E. As the window moves toward a boundary of image memory and crosses that boundary, a wrap-around feature can be implemented for continuous geometric processing without discontinuities caused by image memory boundaries. For example, with reference to Fig 2B; for translation of a window in a particular direction 205K, the window may cross a boundary 205L of image memory 205B, may be wrapped-around to the opposite boundary 205M, may translate in a direction 205K toward the same boundary 205L and again cross the same boundary 205L, may be wrapped around again to the opposite boundary 205M, and so forth. Mosaic elements from the database can be overlayed at the opposite boundary 205M from which the window is moving away. Therefore, an observer viewing the image through the window will view a continuously moving image as the window wraps-around to the new mosaic overlayed on the opposite boundary of image memory. Similarly, rotation and expansion may cause the window to move past a boundary of image memory, which can be accommodated by virtual scrolling.

Window motion, such as horizontal and/or vertical translation, with wrap-around may be characterized as a form of scrolling, horizontal scrolling and/or vertical scrolling. Such wrap-around motion together with overlaying from a database may be characterized as "virtual scrolling". This permits a relatively large image to be implemented with a relatively small image memory by overlaying from the database and by wrapping-around image memory as the window crosses a boundary of image memory.

Wrap-around is discussed with reference to the BASIC PROGRAM LISTING DIS.ASC provided herein. Portions of the window and portions of the image that extend beyond the boundary of the image memory are wrapped-around. For example, a test is made to determine if the pixel coordinate is within the image memory boundary. If it is, the pixel is displayed at the present position. If it is not, the pixel is offset to wrap-around to the opposite boundary. The wrap-around implementation shown in the BASIC PROGRAM LISTING DIS.ASC provided herein is executed independent of whether wrap-around occurs as a result of rotation, translation, compression, or expansion or any combination thereof.

Wrap-around is shown implemented to wrap-around image memory, not database memory or the output window. In an alternate configuration, wrap-around can be implemented for the database memory or the output window in accordance with the teachings for wrap-around in image memory.

Spatial filtering involves weighting and summing of adjacent pixels. However, pixels at the boundary of the memory map may have some of the adjacent pixels wrapped-around. One arrangement for spatial filtering of pixels at the boundary of the memory map is to access adjacent pixels, which were determined to be outside of the memory map, from the opposite side of the memory map in wrap-around fashion. An example will now be provided relative to Fig 3B. In image memory 310E; a pixel at the positive X-boundary 310H of the memory map would access more positive adjacent X-pixels from the pixels at the negative X-boundary 310G of the memory map, a pixel at the negative X-boundary 310G of the memory map would access more negative adjacent X-pixels from the pixels at the positive X-boundary 310H of the memory map, a pixel at the positive Y-boundary 310I of the memory map would access more positive adjacent Y-pixels from the pixels at the negative Y-boundary 310F of the memory map, and a pixel at the negative Y-boundary 310F of the memory map would access more negative Y-pixels from the pixels at the positive Y-boundary 310I of the memory map.

### Clipping

The image processing system can be implemented to automatically clip portions of the image that extend beyond a boundary. For example, in the configuration shown in Fig 2B, when window 205A extends outside of image memory 205B, clipping can be implemented as an alternative to wrap-around and virtual scrolling. The portion of window 205A extending beyond the boundary of image memory 205B can be clipped. Although wrap-around and virtual scrolling may be preferred for many applications; clipping may be pertinent for certain applications, such as applications that do not have a database image to implement virtual scrolling or do not need wrap-around capability.

### Relative Motion

Motion of an image can be defined as being relative to the reference of the observer. For example, for CCW-rotation as viewed by the observer at the viewport, the scanlines are rotated in a CW-angular direction to provide this visual effect. Therefore, the sign associated with the slope parameters may be implemented as negative signs relative to the viewport reference. For example, for CCW rotation in the viewport; it would appear that a positive XP-slope is needed for line direction and a positive XR-slope is needed for CCW-rotation. However, CCW-rotation as viewed by an observer at the viewport can be implemented with CW-rotation of the window about the image memory. Therefore, a positive XP-slope and a negative XR-slope can be used to implement CW-rotation of the window about the image memory to provide CCW-rotation to an observer through the viewport.

### Joystick Controls

Interactive real time operation is demonstrated with the use of joystick controls. The operator can control X-translation, Y-translation, rotation, and expansion/compression with the joysticks. Two joystick configurations have been implemented. For initial experiments, a keyboard joystick was implemented and operation was demonstrated therewith. For subsequent experiments, 2-axis analog joysticks were implemented and operation was demonstrated therewith. Analog joystick implementation includes analog to digital converters (ADCs) implemented on the rear end board BR1 (Figs 6T to 6V) and includes software control set forth in the BASIC PROGRAM LISTING DIS.ASC provided herein.

Keyboard joystick controls will now be discussed. Keyboard inputs to the supervisory processor can be used to provide joystick-type controls. The display program can be implemented to be responsive to terminal keyboard key-1 through key-8 representing +X, -X, -Y, +Y, expansion, compression, +angle, and -angle commands; respectively. The +X, -X, -Y, and +Y commands are translation acceleration commands; incrementing and decrementing the X-delta and Y-delta velocity parameters in response to key actuation. The +angle and -angle commands are rotation acceleration commands; incrementing and decrementing the angle delta velocity parameter in response to key actuation. The expansion and compression commands are scale factor products, decreasing and increasing the image scale factor respectively by a predetermined factor in response to key actuation.

Joystick key signals are multiplexed into a single serial port used for the keyboard input to the computer. Therefore, a single key command is executed at a time. Sequential manipulation of the keys permits combinations of X-translation, Y-translation, X-rotation, and expansion/compression to be performed in accordance with the previously generated sequence of commands. Continued depression of a command key causes a more rapid command rate for that function, such as a high rate slew of the command parameter. Stroke depression of a key, without continued depression of that key, provides a precisely controllable incremental command. For example, 3-depressions of the +X switch results in +X motion of a commanded amount and then 3-depressions of the -X switch cancels X-motion and results in zero X-axis motion.

F Analog joystick control will now be discussed. The analog joysticks include 2-separate joysticks, each having 2-orthogonal axes of control. Each axis of control is implemented with an analog potentiometer in the joystick and an ADC on the rear-end board BR1 (Figs 6T to 6V). The software initiates converter operation and then, after a period of time, reads the output of the converters as indicative of the magnitude of each of the joysticks. The +X, -X, +Y, and -Y commands are translation velocity commands; updating the X and Y position parameters in response to joystick commands. The +angle and -angle commands are rotation velocity commands; updating the angle parameter in response to the joystick commands. The expansion and compression commands are scale factor products, updating the image scale factor by a factor related to the magnitude of the joystick

command.

Manipulation of joysticks, both keyboard emulated joysticks and actual linear joysticks, controls geometric processor to be continually changed in real time under operator control. For example, positive rotation can be initiated, can be increased in rate, can be decreased in rate, and can be stopped by manipulation of the +angle and -angle switches used to emulate joysticks. Also, positive and negative X-translation and Y-translation can be individually and in combination increased in rate, decreased in rate, and stopped by manipulation of the related switches used to emulate joysticks. Similarly, expansion and compression can be commanded to increase image size and to decrease image size.

Experimental System Video Tape

Video tape recordings were made of images generated with the experimental system moving under joystick control. Video tapes have been filed in the disclosure documents

1. Disclosure Document No. 126,823 filed on March 19, 1984.
2. Disclosure Document No. 126,825 filed on April 26, 1984.
3. Disclosure Document No. 127,956 filed on June 4, 1984.

which are herein incorporated by reference.

A video camera was used to record images displayed on the color monitor of the display processor. The good resolution and brilliant colors of the display processor display could not be recorded in full detail or color because of the lower resolution and lower quality of the video camera and video tape. However, the recorded quality is adequate to demonstrate actual reduction to practice of dynamic real time operation under operator control with simultaneous rotation, translation, and expansion/compression.

The video tape filed as Disclosure Document No. 126,825 will now be discussed, which is generally representative of the other two video tape disclosure documents referenced above. Various images were overlayed on a blue background using a version of the BASIC PROGRAM LISTING LD.ASC provided herein. These images were manipulated under joystick control to provide dynamically moving images. Motion is shown to be smooth and continuous in real time and images are shown wrapping around. The more important images are images that are <sup>1</sup><sub>^</sub> inbetween unity size and medium expansion; because images outside of this range have aliasing effects, resolution effects, and other effects that detract from

appearance of the demonstration. However, presentation of these effects illustrates characteristics of the demonstration system and shows extremes of operating parameters.

As images are expanded to large size, aliasing effects and discrete motion effects become more apparent as a function of the degree of expansion. This results from expanded images being portrayed with lower significant bits of the slope registers and hence utilizing less of the resolution of the slope registers than available for more compressed images. Also, such expansion exaggerates pixel resolution in image memory; presenting exaggerated aliasing effects, such as staircasing.

As images are compressed to small size relative to image features, such as compressed bar patterns to one pixel width per bar or more compressed therefrom; exaggerated aliasing effects are introduced, such as distorted patterns.

Simultaneous motion having combinations of X-translation, Y-translation, rotation, and expansion/compression are shown in real time with smooth continuous motion. Speed of motion is shown ranging from very slow motion controlled by small displacements of the joysticks through fast motion controlled by large displacements of the joysticks.

Blanking, which has been implemented with blanking signals to the DACs on the rear-end board BR1, has been modified to show the images in unblanked form. The effects that are blanked during usual operation are visible here, including distorted image effects around the periphery of the viewport, characterized by the distortions at the far right of the viewport.

Expansion and compression are usually limited to a range consistent with the nature of the images to minimize the effects of aliasing. However, in order to demonstrate these effects, expansion and compression have been extended beyond these limits. The effects of compressive aliasing, such as undersampling, can be seen when most images are compressed. For the single pixel wide bar test patterns; such as the single pixel wide red, green, blue, and black bar test pattern; the effects of compressive aliasing can be seen for slight compressed and even for unity size. The effects of expansion aliasing, such as pixel replication and reduced spatial resolution, can be seen in the highly expanded images. Expansion aliasing manifests itself with larger staircasing, caused by expanding the image memory spatial resolution; reduced resolution motion in larger steps, caused by operating with small fractional slopes in the low resolution portions of the slope registers; and jitter as a highly expanded object is rotated, expanded/compressed, caused by position roundoff to pixel resolution being magnified as the pixels are magnified to larger size.

For the configuration of the experimental system used to generate the video tape filed as Disclosure Document No. 126,825; X-translation and Y-translation are referenced to the image memory, not to the viewport. Therefore, rotation of the viewport causes rotation relative to the X and Y joystick controls that are referenced to the image being rotated. In subsequent configurations set forth in the BASIC PROGRAM LISTING DIS.ASC, joystick controls are resolved into viewport coordinates so that the joystick controls remain fixed to the viewport while the

image is being rotated. Representative equations are as follows.

$$JSXR = JSX * \cos(AR) + JSY * \sin(AR)$$

$$JSYR = JSX * \sin(AR) + JSY * \cos(AR)$$

The color monitor is a medium resolution commercial quality monitor having good spatial frequency response and having brilliant colors. The video camera and video tape recorder used to record these images are consumer quality units and hence result in degradation of the images. However, the image quality is sufficient to demonstrate an actual reduction to practice of the features implemented in the demonstration system. The full quality of the images can be obtained by one skilled in the art by constructing the demonstration system according to the teachings herein and by operating the demonstration system in accordance with the teachings herein.

The images portrayed on the video tape provide demonstration of many of the features discussed herein in various combinations and with many different scenarios to establish actual reduction to practice over a wide range of demonstratable scenarios. For example, the demonstrations provide rotational motion for many different conditions of scaling, X-position, and Y-position and provide rotational motion in combination with expansion and compression; in combination with X-translation; in combination with Y-translation; in combination with expansion and X-translation; in combination with compression and X-translation; in combination with expansion, X-translation, and Y-translation; and with other combinations of effects.

Also, the demonstrations provide translational motion for many different conditions of scaling, X-position, Y-position, and rotation and provide translational motion in combination with expansion and compression; in combination with rotation, in combination with expansion and rotation; in combination with compression and rotation; and with other combinations of effects.

Further, the demonstrations provide expansion for many different conditions of scaling, X-position, Y-position, and rotation and provide expansion in combination with rotation; in combination with X-translation; in combination with Y-translation; in combination with rotation and X-translation; in combination with rotation and Y-translation; in combination with rotation, X-translation, and Y-translation; and with other combinations of effects.

Yet further, the demonstrations provide compression for many different conditions of scaling, X-position, and Y-position and provide compression motion in combination with rotation; in combination with X-translation; in combination with Y-translation; in combination with rotation and X-translation; in combination with rotation and Y-translation; in combination with rotation, X-translation, and Y-translation; and with other combinations of effects.

The images portrayed on the video tape demonstrate many sequences of operation. For example, the joysticks were manipulated in a circular fashion to provide sequences of plus and minus X and Y motion and to provide sequences of expansion and compression and plus and minus rotation. The rapid response to the joystick controls and the sequence of operation can be seen from the video tape playback.

### Other Geometric Processor Configurations

Various configurations of the geometric processor of the present invention have been discussed with reference to Fig 2 above to illustrate how the various features and devices of the geometric processor of the present invention can be used to implement a system. These configurations are illustrative of a large number of other configurations that can be implemented from the teachings herein.

The different geometric processor features disclosed herein can be used in different combinations and with different interconnections, illustrated by the exemplary combinations and interconnections discussed herein but not limited to the exemplary combinations and interconnections discussed herein.

The geometric processor configuration discussed above has been discussed in the configuration of a window configuration. This window configuration is illustrative of other configurations that do not use a window architecture. For example, a plurality of address generators can be used to address a first memory for accessing of pixels and for writing into a second memory in transformed form to implement geometric processing. The second memory can be a buffer memory, a scratch pad memory, a refresh memory, or other memory. The window configuration has been discussed for implementing geometric processing by controlling the window parameters of initial point coordinate and slope parameters. Alternately, window geometric processing can be implemented by controlling other window parameters; such as the scanline start-point and end-point, angle of rotation, expansion and compression scale factor, and other geometric parameters.

The geometric processor configuration discussed above has been discussed in the configuration of address counters counting along scanlines having the commanded position, rotation, expansion or compression, warping, and other characteristics. Alternately, geometric processing can be implemented with whole number processing elements; such as matrix processing elements, incremental processing elements, geometric processing elements, and other processing elements.

The geometric processing configuration discussed above has been discussed in the configuration of a stored program processor implementing once per field processing and a hardwired logical processor implementing line and pixel processing. Alternately, the geometric processor can be implemented with stored program processors for the field, line, and pixel processing; can be implemented with hardwired logical processors for the field, line, and pixel processing; can be implemented with stored program processors for the field and line processing and with hardwired logical processors for the pixel processing; and with other implementations thereof. Processors can include digital stored program computers, digital incremental processors, digital hardwired logic processors, analog processors, hybrid (analog and digital) processors, and different combinations thereof.

The geometric processing configuration discussed herein has been discussed in the configuration of an interlaced raster scan processor. Alternately, a non-interlaced scan configuration can be implemented; such as by making XIP2 and YIP2 equal to XIP1 and YIP1 respectively so that both fields are superimposed instead of

being interlaced or by deleting the second field processing and generating the first field at half the line spacing of the interlaced scan configuration. Also, other scan arrangements can be used in place of the raster scan; such as the Palmer scan, PPI scan, A-scan, B-scan, <sup>1</sup> caligraphic vector generation, and others.

The geometric processing configuration discussed above has been discussed in the configuration of a synch pulse controlled processor. Alternately, geometric processing can be implemented to be self-controlled, such as by generating geometric processing control signals for synchronizing the display monitor to the geometric processor; can be implemented to be asynchronous, such as by <sup>1</sup> asynchronously storing output information in an auxiliary memory and then synchronously outputting the stored output information to a display; and can be implemented to operate in other synchronous and asynchronous forms.

The geometric processing configuration discussed above has been discussed in the configuration of a multiple loop geometric processor having an outer field loop, middle scanline loop, and inner pixel loop. Alternately, geometric processing can be implemented with a different loop arrangement; such as with an outer frame loop instead of the outer field loop, or without an outer field or frame loop and having processing determined on a scanline by scanline basis, or with other configurations. Alternately, geometric processing can be implemented for generating images on a pixel-by-pixel basis without any loops.

The geometric processor configurations discussed herein can be used with the other elements discussed herein, such as the spatial filter. However, the geometric processor can be used

separate from other devices discussed herein. For example, the geometric processor can be used without spatial processing or with different types of spatial processing than discussed herein.

The geometric processor configurations discussed herein have been discussed in conjunction with other features of the present invention; such as memory features, buffer features, and application features. However, the geometric processor of the present invention provides important advantages that can be used in conjunction with conventional devices; such as conventional memories, conventional buffers, and conventional applications.

The geometric processor configurations discussed herein have been discussed for an image processing system. However, many of the geometric processor teachings can be utilized for other than image processing systems, such as graphic display systems.

GRAPHICS PROCESSOR

A graphics processor architecture can be implemented with a address generator and control logic generating graphics vectors for storing into image memory. Image memory can then be scanned out, such as in a raster scan form to refresh a display. In one configuration, graphics vectors can be written into image memory on an offline basis and can be used to refresh the display on an online basis. Alternately, graphics vectors can be written into image memory on an online basis time shared with refreshing of the display on an online basis.

One arrangement of the graphics system of the present invention is shown in Fig 1P. Supervisory processor 115A loads graphics commands into address generators 115B. Address generators 115B generate addresses of graphics vectors for loading into image memory 115C and for raster scanning image memory 115C. The raster scan addresses scan-out the image in image memory 115C through the CRT interface 115D to refresh CRT 115E.

An experimental system has been constructed to demonstrate operation of the graphics display capability. The arrangement shown in Fig 1 has been implemented in hardware for refreshing the display in real time. A program, such as the BASIC PROGRAM LISTING GRAPH.ASC, can be used to control that experimental hardware for refreshing the display. In this experimental system, the graphics vectors are loaded in an offline manner with the LD.ASC Basic program set forth in the BASIC PROGRAM LISTING LD.ASC herein; emulating hardware loading of graphics vectors in an online manner. In this experimental system, graphics operation is initiated each frame with supervisory processor 115A and

hardware refresh is performed with address generators 115B and image memory 115C.

In a hardware configuration, graphics vector generation can be performed in real time using the software emulated vector generation capability implemented in hardware form. In one hardware configuration, graphic vectors can be generated coterminously with refresh, such as with one set of address generators (i.e., the XR-address generator and the YR-address generator shown in Figs 6Q and 6R) generating graphics vectors into image memory while a second set of address generators (i.e., the XP-address generator and the YP-address generator shown in Figs 6O and 6P) are generating the raster scan addresses for scanning-out image memory for display. In this configuration, image memory can be implemented as a dual-ported image memory for simultaneously loading vectors into image memory and scanning-out image memory. In an alternate hardware configuration, graphic vectors can be generated and loaded into image memory during the vertical sync pulse period when the raster scan is blanked; time sharing the logic and memory between raster scanout and graphics generation. In this configuration, during the vertical sync period, the address generators can generate graphic vector addresses for loading the graphic vectors into image memory and, after the vertical sync period, the address generators can generate the raster scan addresses for scanning-out image memory for display.

The address generators can be used to generate graphic vectors and windows. For example, the LD.ASC program set forth in the BASIC PROGRAM LISTING LD.ASC herein has been used to load graphic vectors into image memory. This is achieved by using the address generators to generate the addresses of a vector and by strobing the color intensity of the vector into image memory.

Periods of time exist when the address generators are in a stand-by condition. For example, in a configuration where the address generators are scanning-out image memory to refresh a display; the address generators may not be used during the vertical blanking period and therefore may be available for graphic generation. Also, in a configuration where the address generators are not used during the horizontal blanking period, the address generators and therefore may be available for graphic generation during the horizontal blanking period. For example, a vertical blanking period of 1-millisecond will permit the address generators to draw about 5,000-graphic vector pixels operating at a 5-MHz pixel rate. Consequently, a meaningful number of graphic vector pixels can be generated during standby periods, permitting time sharing of the address generators for both, scanning-out an image to refresh a display and graphic vector generation.

A vector memory can be implemented to store parameters associated with the vectors to be generated. Vector memory can be loaded from various sources, such as from the supervisory processor that initializes the address generators, from a host processor, or from other sources. The vector memory can contain the start point coordinates and the vector deltas for the address generators and a quantity parameter or distance-to-go (DTG)

parameter related to the quantity of vector steps to be generated for the particular vector. During image processing standby periods, graphic vector parameters can be loaded from the vector memory for generating the vectors with the address generators, similar to that performed with the LD.ASC program. After various standby periods, such as the horizontal and vertical synchronization periods; the address generators can be reinitialized; thereby overcoming the need to buffer scanout parameters. However, if the address generators will not be reinitialized following vector generation, it may be necessary to buffer the scanout address parameters in a buffer memory for reloading the pixel address generators after vector generation.

In the LD.ASC program, the number of steps for a vector are counted under program control in the supervisory processor. In a hardwired implementation, the number of steps for a vector can be counted with a hardware counter circuit. For example, the quantity or DTG parameter from the vector memory can be loaded into a 74LS169 counter as a parallel load parameter and the counter can be decremented in the count-down mode for each pixel step during vector generation. Generation of the vector can be terminated by detecting the underflow signal from the counter at the zero count.

Loading of the address generators from the vector memory can be performed in a manner similar to loading the address generators from the supervisory processor, as shown in the LD.ASC program listing herein and as discussed relative to the supervisory processor interface herein. Setting of the vector

color intensity from the vector memory can be performed in a manner similar to setting of the vector color intensity from the supervisory processor in the LD.ASC program. Selecting of the write-mode for the image memory can be performed in a manner similar to setting of the write-mode with the load command signal DOA6 by the supervisory processor in the LD.ASC program.

Window generation can be implemented with parameters for a plurality of images stored in a window buffer memory and selected as the address generators scan across window boundaries during scanout and refresh of the CRT monitor. When the address generators cross window boundaries, the previous display parameters can be buffered in the buffer memory and the display parameters associated with the new image can be loaded from the window buffer memory into the address generators. Loading of display parameters associated with the new image from the window buffer memory can be accomplished as discussed above for loading of vector parameters during graphic vector generation. Storing of display parameters associated with the prior image into the window buffer memory can be accomplished by reversing the vector generation loading operation to obtain a window generation store operation.

SPATIAL FILTERING

## Description Of Figs 5A To 5C

9pm  
10/19/84

Display systems can be implemented with spatial filters for anti-aliasing, pattern recognition, enhancement, and other purposes. A spatial filter arrangement will now be discussed with reference to Figs 5A to 5C.

Fig 5A shows an arrangement of a display system. Address generator 520A generates pixel addresses to access a plurality of pixels, such as a 9-pixel kernel 520H, from image memory 520B. Pixel information can be latched in registers to provide parallel pixel words or can be accessed sequentially as provided with the BASIC PROGRAM LISTING <sup>FTR</sup>GRAPH.ASC herein. Weight table 520C supplies a plurality of kernel weights appropriate to spatial filtering of the pixel kernel, such as a kernel of 9-weights 520I, from weight table 520C. Weight information can be latched in registers or in the weight table to provide parallel pixel words or can be accessed sequentially as provided with the BASIC PROGRAM LISTING <sup>FTR</sup>GRAPH.ASC herein. The pixel intensities I0 to I8 are each applied to a corresponding multiplier 520E and the weights W0 to W8 are each applied to a corresponding multiplier 520E for multiplying the corresponding intensity and weight together to generate product signals 520J. Product signals 520J are summed together with summer 520F to generated a weighted and mixed pixel intensity, which is converted to analog signal form with DAC 520G to excite a CRT display.

The arrangement discussed with reference to Fig 5A is representative of a single color channel, such a single channel of a multiple color pixel; i.e., a red, green, or blue channel; and such as a monochromatic single channel. Intensity information

INT and weight information WT can be input to multipliers 521A for weighting the pixel intensities, which in turn can be input to adders 521B and 521D for generating weighted and summed signal 521D. Three channels of the arrangement discussed with reference to Fig 5B can be combined to provide a 3-channel color spatial filter. For example, as shown in Fig 5C, 3-channels of intensity and weight information 521E are processed with sum-of-the-products logic 521F to generate 3-channels of signals 521G; such as red, green, and blue signals 521D.

The sum-of-the-products processing discussed above can be implemented with commercially available integrated circuit components, such as multiplier chips and adder chips. For example, multiplier chips are manufactured by TRW and adder chips are manufactured by Texas Instruments.

Description of Fig 7D and the FTR.ASC Listing

A filter processor can be emulated in software to illustrate operation of a hardware configuration. One filter processor configuration is shown in the BASIC PROGRAM LISTING FTR.ASC provided herewith and is shown in flow diagram form in Fig 7D. This program provides for accessing of pixels from the database, filtering the pixels, and storing the filtered pixels in another file in the database. The unfiltered file is defined as UNFILTER.ASC and the filtered file is defined as FILTERED.ASC. In this configuration, the database is constructed as 8-stripes, each stripe having 64-lines, and each line having 512-pixels. This comprises an image 512-lines by 512-pixels per line. The image database is constructed as a packed byte per pixel with 128-pixels stored in a sector on the disk. The Basic program FTR.ASC accesses the binary file as a Basic random file and unpacks the pixels for filtering under the Basic program. For convenience of implementation, 2-pixels at a time are unpacked for filtering and 2-pixels at a time are packed after filtering.

The filtering program will now be discussed with reference to the flow diagram in Fig 7D. The program is implemented with a plurality of loops. The outer loop KRLP1 defines the vertical coordinate of the pixel kernel. The next inner loop KPLP1 defines the horizontal coordinate of the pixel kernel. The next inner loop OUTLP1 defines the vertical coordinate of the pixel line in the kernel. The innermost loop INLP1 defines the horizontal coordinate of the pixel pair on the pixel line in the kernel. The program processes 3-pixels per line in the kernel with 2-iterations through the inner loop INLP1, then processes 3-

lines in the kernel with 3-iterations through the next outer loop OUTLP1, then processes 512-kernels per line with 512-iterations through the next outer loop KPLP1, and then processes 64-lines of kernels per stripe with 64-iterations through the outer loop KRLP1.

The flow diagram (Fig 7D) is drawn with elements having descriptive labels that correspond with the program operations. The line numbers associated with these operations are listed above the elements for convenient cross-referencing to the Basic program FTR.ASC. Reference numerals 760A to 760AB are shown with connecting lines to the elements for convenient cross-referencing in the description.

Execution commences by generating initial conditions in element 760A. Operation then proceeds to the outer loop KRLP1 760B to address the first kernel position at the top edge of the kernel array and then proceeds to the next inner loop KPLP1 760C to address the first kernel position at the left edge of the kernel line. Operation then proceeds to element 760D to initialize the KPLP1 loop by generating the initial conditions XB, YB, XE, YE, and BDC. Operation then proceeds to the next inner loop OUTLP1 760E to address the top line of pixels in the kernel. Operation then proceeds to element 760F to access a database record corresponding to the top line of pixels in the kernel. Operation then proceeds to the inner loop INLP1 760G to address the pair of pixels at the leftmost side of the kernel in the selected line.

Operation then proceeds to element 760H to access and unpack

a pixel pair and then to element 760X to select the weights for the selected pixel line. Operation then proceeds to element 760Y to test the S1-flag for processing of either the first pixel pair or the second pixel pair. If the S1-flag is 0-set, indicative of processing of the first pixel pair; operation proceeds along the 0-path to element 760AA to weight the first pair of pixels and to sum the weighted pixels in a partial summation. Operation then proceeds to element 760AB to 1-set the S1-flag, indicative of the need to process the second pixel pair on the next iteration. In element 760Y, if the S1-flag is 1-set, indicative of processing the second pixel pair; operation proceeds along the 1-path to element 760Z to weight the second pair of pixels and to sum these weighted pixels with the partial sum previously generated with the first pair of weighted pixels. Only the first pixel of the second pair of pixels is processed, where a 3-pixel line for a kernel is implemented with both pixels of the first pixel pair and the first pixel of the second pixel pair.

Operation then proceeds to element 760J to define the next pixel pair on the selected line of pixels and then to element 760K to determine if the last pixel pair per pixel line in the kernel has been processed. If the last pixel pair of the kernel has not been processed, operation branches along the Y-path looping back in the INLP1 loop to access the next pixel pair on the selected line of pixels. If the next pixel pair of the kernel has been processed, operation branches along the N-path to exit the INLP1 loop, then to element 760L to 0-set the S1-flag, and then to element 760M to determine if the last pixel line of the kernel has been processed. If the last pixel line of the

kernel has not been processed, operation branches along the Y-path looping back in the OUTLP1 loop to access the next line of pixels in the kernel and to iterate through the INLP1 loop to access the kernel pixels on that line and to filter these kernel pixels. If the last pixel line of the kernel has been processed, operation branches along the N-path to exit the OUTLP1 loop and then to element 760N to test the S2-flag.

The S2-flag identifies whether the filtered pixel is the first or second pixel in a pixel pair. If the pixel is the first pixel in a pixel pair identified by the S2-flag being 0-set, operation branches along the 0-path to element 760P to 1-set the S2-flag indicative of the first pixel pair being processed and then to element 760Q to scale the filtered pixel and to buffer the filtered pixel. If the pixel is the second pixel in a pixel pair identified by the S2-flag being 1-set, operation branches along the 1-path from element 760N to element 760R to scale a second filtered pixel and to element 760S to store the filtered pixel pair in the database and to 0-set the S2-flag as indicative of the first pixel of the next pixel pair pending processing.

Operation then proceeds to element 760U to determine if the last kernel per kernel line has been processed. If the last kernel per line has not been processed, operation branches along the Y-path looping back in the KPLP1 loop to process the next kernel along the line of kernels. If the last kernel per line has been processed, operation branches along the N-path to element 760V to determine if the last line of kernels has been processed. If the last line of kernels has not been processed,

operation branches along the Y-path looping back in the KRLP1 loop to process the next line of kernels. If the last line of kernels has been processed, operation branches along the N-path to element 760W to exit the filtering of the stripe.

## Filtering of Images

Image processing can be performed on digitized images stored in the database. A stored image can be loaded from the database into image memory for image processing online with a hardware-implemented image processor. A stored image can be preprocessed, such as offline in the database and such as online with a preprocessor. Offline processing can be performed during database generation, such as filtering database images before storing the images in the database. Online preprocessing can be performed during system operation, such as by accessing database images and preprocessing the accessed database images with compression and filtering, prior to loading into image memory. Offline preprocessing can be performed for filtering of images, which is illustrative of other forms of offline preprocessing and is also illustrative of online preprocessing. For example, other forms of offline preprocessing includes data compression, image enhancement, and other forms of filtering. Also, other forms of online preprocessing includes data decompression, image expansion, image compression, and filtering.

Offline filtering and the other preprocessing operations illustrated therewith is shown in the BASIC PROGRAM LISTING *FTR.FLT.ASC* provided herein.

For this example, the image is constructed in database memory as 512-lines and 512-pixels per line. Each pixel is implemented with an 8-bit byte having the least significant 3-bits representing green intensity, the next more significant 2-bits representing red intensity, the next more significant 2-bits representing blue intensity, and the most significant bit being

presently unused. The image is divided into 8-subimages; where each subimage contains 64-lines in a stripe. The stripes are identified as stripe-0 to stripe-7. For example, the lake image has the topmost stripe stored as the LAKE50.BIN file, the next lower stripe stored as the LAKE51.BIN file, the other stripes stored in the ascending LAKE5n.BIN files and the lowest stripe stored in the LAKE57.BIN file. The filter program accesses each stripe in sequence, filters the stripe, buffers the stripe as the FILTERED.BIN file, and stores the stripe as the FLAKE5n.BIN file. The filtering operation accesses a kernel of pixels surrounding the center pixel and consequently overlaps stripes for the first line and the last line in a stripe. Therefore, filtering of each stripe is implemented by accessing the particular stripe as the PRESENT.BIN file, accessing the previous stripe as the PRIOR.BIN file, and accessing the subsequent stripe as the NEXT.BIN file.

In this example, filtering is implemented with a 9-pixel kernel organized in a 3-pixel by 3-pixel array. This filtering is performed by multiplying each of 9-kernel pixels by the corresponding one of 9-kernel weights, summing the weighted pixel intensities, and then scaling to normalize the filtered intensity. This processing is performed for each of the 3-color components of the kernel pixels. Weights can be assigned to the particular kernel positions, selectable under operator control with the menu. Kernel weights No. 1 and kernel weights No. 2 are shaded, where the weights are decreasing from the kernel center towards the kernel periphery. Kernel weights No. 3 are unshaded. Additional kernel weights can be defined by the operator under menu contr<sup>ol</sup>/

## Spatial Filtering With Geometric Processing

Spatial filtering will now be discussed, where many of the features can be implemented with the arrangement shown in Fig 1A having spatial filtering modules and data paths for feed forward and feed back spatial filtering. Spatial filtering is discussed herein with reference to the BASIC PROGRAM LISTING FTR.ASC provided herewith and discussed with reference to Fig 7D.

Spatial filtering can be implemented with a multiple pixel kernel, such as a 3-pixel by 3-pixel kernel for implementing weighting of intensities of adjacent pixels and summing to obtain an output pixel intensity. The adjacent pixel intensities can be loaded into kernel registers and each pixel intensity can be multiplied by an appropriate weight, such as from a weight memory, and the weighted pixel intensities can be summed together; as discussed with reference to Figs 5A to 5D and 7D herein. The resulting signal represents a smooth pixel intensity. Other spatial filtering arrangements can also be used. Filtered images can be used to provide improved displays, can be used for pattern recognition, can be used for image enhancement, and can be further processed in feed-forward and feed-back configurations.

Spatial filtering can be implemented as pre-filtering, post-filtering, and combinations of pre-filtering and post-filtering. Pre-filtering may be considered to be filtering that is performed before geometric processing. Post-filtering may be considered to be filtering performed after geometric processing. For example, pre-filtering can be performed to reduce the spatial frequencies prior to geometric processing in order to reduce aliasing. Post-

filtering can be performed after geometric processing to reduce aliasing and to provide smoothing.

Separate filters, a pre-processing filter and a post-processing filter, can be implemented for concurrent pre-filtering and post-filtering. Alternately, a single filter can be shared between pre-filtering and post-filtering operations. For example, a post-filter can be implemented to filter the geometrically processed information for outputting to the display monitor. Alternately, the post-filter can be implemented to filter an image that has either been geometrically processed, such as to output to a CRT monitor, or has not as yet been geometrically processed, such as for recirculation to an image memory. This image memory can be the source image memory that originally contained the unfiltered image for storing the filtered image without preserving the unfiltered image or can be another image memory for storing the filtered image while preserving the unfiltered image. The filtered image can then be geometrically processed as a pre-filtered image for output to the display monitor without post-filtering or alternately for post-filtering prior to outputting to the display monitor.

Pre-filtering and post-filtering can each be performed progressively, such as discussed herein for progressive compression. Progressive filtering can be used for the purposes of progressive compression, progressive low pass filtering, and progressive feature enhancement. The progressive filtering can be performed with the same weights for each iteration or, alternately, with different weights for different iterations.

For example, progressive compression and progressive low pass filtering can be performed with the same weights for each iteration, such as to progressively compress to a large compression magnitude and to progressively low pass filter to a low spatial frequency, or can be performed with different weights for each iteration, such as to enhance different feature characteristics for each iteration with different sets of weights.

Progressive filtering can be performed in an off-line manner and in an on-line manner. Off-line progressive filtering can be filtering without display of the intermediate progressive filtered information. On-line progressive filtering can be filtering where the filtered information is displayed as it is iteratively filtered. Off-line progressive filtering can be implemented by progressively filtering an image, such as to obtain high degrees of compression or high degrees of low pass filtering; where the progressively filtered image is continually recirculated to an image memory without being output to the display monitor until the progressive filtering has been completed. On-line progressive filtering can be implemented by progressively filtering the image and displaying the filtered image, such as for each iteration; while the filtered image is also recirculated back to the image memory for subsequent iterations.

On-line progressive filtering will now be discussed. Many applications involve images that change continuously, where it is desirable to display the continuous changes; such as continuous motion of the image. For such applications, on-line progressive

filtering may be particularly efficient. The image can be filtered in a pre-filtering or post-filtering manner, can be geometrically processed, and can be displayed and recirculated for subsequent progressive filtering iterations. This permits simultaneous operation of both, the progressive filter and the displaying of the image as it is progressively filtered.

Off-line progressive filtering will now be discussed. An off-line filtering arrangement can be used in conjunction with an on-line filter arrangement. The off-line filtering arrangement can provide progressive filtering, such as in conjunction with a buffer memory; while the image is being geometrically processed and may also be pre-filtered or post-filtered in an on-line manner for display. After progressive off-line filtering has been completed, the off-line progressively filtered image, such as stored in a buffer memory, can be switched to an on-line mode for display of the image that was progressively filtered in an off-line manner. Such off-line progressive filtering may be considered to be a pre-filter that provides an image for display which may not be continuous in that it is displayed after progressive filtering instead of during progressive filtering.

Various types of spatial processing can be provided, such as pre-processing and post-processing. Pre-processing can perform various operations, such as low pass filtering an image to prevent aliasing due to spatial frequencies in the image that are greater than half of the sampling frequency in accordance with the well known Nyquist criterion. Post-processing can perform various operations, such as overcoming staircasing and pixel resolution considerations. Pre-processing can reduce spatial frequencies so that aliasing effects are not introduced into the image when the image in image memory is sampled with the image processor for outputting to the display. The sampling frequency can be determined by the scale factor of a display processor, such as for expansion and compression. For expansion, the spatial frequencies are lower and hence aliasing is reduced. For compression, the spatial frequencies are greater and hence aliasing considerations are more important. Therefore, as the image is expanded, the anti-aliasing filter related spatial frequency may be increased to provide greater detail and, as the image is compressed, the anti-aliasing filter related spatial frequency may be reduced to reduce the detail.

For optimum detail, the related frequency of the anti-aliasing filter may be continually adjusted to optimize the amount of detail. As the image is compressed, the related frequency of the anti-aliasing filter can be reduced as a function of compression, continually reducing the amount of detail in the image. As the image is expanded; the related frequency of the anti-aliasing filter can be increased to increase the amount of detail.

The amount of detail in a previously anti-alias filtered image may not be increased by re-filtering with a higher effective spatial frequency anti-aliasing filter. This is because the detail has already been filtered out to the level of the lower effective spatial frequency of the anti-aliasing filter and hence cannot be put back in merely by increasing the effective spatial frequency of the anti-aliasing filter. One way to increase the detail of the image is to reload the image from a higher resolution storage, such as a front end buffer memory or the database memory, which has not been filtered to a lower effective spatial frequency than is now desired.

Post-processing can be performed on a geometrically processed image that has been pre-processed before geometric processing. This post-processing is discussed herein in the form of a triple line buffer generating a 9-pixel kernel and processing the 9-pixel kernel in accordance with a set of weights using sum-of-the-product processing logic. This provides various capabilities, as previously discussed; such as increasing intensity resolution with processing gain enhancement, increasing spatial resolution to subpixel resolution, smoothing of staircasing effects, smoothing of motion, and other such advantages.

Once an image has been pre-filtered for a particular level of expansion or compression, it may not need to be pre-filtered again until the geometric processor needs to change the level of expansion or compression. In one configuration previously discussed, an image is pre-compressed and pre-filtered for that

level of compression and stored in various pre-compressed and pre-filtered stages in database memory. In such an arrangement, the appropriate stage of pre-compression and pre-filtering consistent with the geometric processor conditions can be accessed from database memory and stored in an input buffer memory. This image can be accessed from the input buffer for loading into image memory for geometric processing. As long as expansion and compression processing is within the range or this pre-compressed and pre-filtered image stage, the image stored in the buffer memory can be pre-processed for anti-aliasing consistent with the level of expansion of compression and loaded into the image memory for geometric processing. Also, as long as the image in the buffer memory is consistent with the translational conditions of the geometric processor, scrolling of the image memory can be satisfied by loading the image memory from the buffer memory as scrolling moves beyond an image memory boundary with the previously discussed virtual scrolling and wrap-around capability.

When the geometric processor condition exceeds either the range of expansion or compression associated with the pre-compressed and pre-filtered image stored in the buffer memory or exceeds the translational capabilities of the image stored in the buffer memory, additional image information can be accessed from the database memory to accomodate these geometric processor conditions. For example, if the range of expansion and compression of the image in buffer memory are exceeded, addition image memory having the new range of expansion or compression can be accessed from the database memory to accomodate the new range.

Similarly, if the translation of the image exceeds the image stored in the buffer memory, additional image information can be accessed from image memory for overlaying in the buffer memory to extend the translation capability of the geometric processor in that direction.

In view of the above, operation can proceed as follows. Initially, a portion of an image having the appropriate scale range and the appropriate translational position can be accessed from database memory and loaded into the buffer memory. A portion of the image in the buffer memory can be accessed for pre-processing to the intermediate scale level to be provided with the geometric processor and can be loaded into image memory for processing with the geometric processor. The geometric processor can process the image in image memory, which can then be post-processed with the post-processor and displayed on the CRT monitor.

As the geometric processor is commanded to translate, the geometric processor will translate through the image in image memory, outputting the image for post-processing and display. As the translation approaches a boundary of the image memory, additional image information can be loaded from the buffer memory and overlaid into image memory behind the translating window. As the information being accessed from the buffer memory approaches a boundary of the buffer memory, additional image information can be accessed from the database and overlaid into the buffer memory behind the direction of translation of image memory.

As the geometric processor is commanded to expand, the geometric processor will compress the window in image memory, outputting the expanded image (compressed window) for post-processing and display. As the expanding image (compressing window) expands, the resolution of the image decreases. Pre-expansion can be provided by loading expanded higher detail image information into image memory; such as from buffer memory or database memory.

As the geometric processor is commanded to compress, the geometric processor will expand the window in image memory, outputting the compressed image (expanded window) for post-processing and display. As the expanding window approaches a boundary of the image memory, additional image information can be loaded from the buffer memory and overlayed into image memory to maintain the expanding window within image memory. As the expanding window approaches to within a portion of image memory size, the image in image memory can be pre-compressed and the window can be reduced in size consistent therewith to maintain the window within a safe size within the boundaries of image memory. Pre-compression can be provided by compressing the image in image memory and loading additional image information to fill the vacated portions of image memory, compressing the image in buffer memory and loading the more compressed image into image memory, loading a more compressed image from database memory, or other arrangements. As the information being accessed from the buffer memory approaches a boundary of the buffer memory, additional image information can be accessed from the database and overlayed into the buffer memory consistent with the window

position in image memory.

Therefore, the image memory is updated with the more appropriate portions of the image from the buffer memory and the buffer memory is updated with the more important portions of the image from the database; where the database stores the full image, the buffer memory stores a subset of this full image, and the image memory stores a smaller subset of this image.

MEMORY ARCHITECTURE

### General

The memory architecture of the present invention has important advantages in implementing digital systems. It is applicable to special purpose systems; such as display systems, array processors, and pipeline processors; and is applicable to general purpose systems; such as general purpose digital computers. It incorporates various features that may be used individually or in combinations to enhance performance and efficiency. One feature provides for accessing of memory at a relatively slow addressing rate and at a relatively fast scanout rate. Another feature provides a buffer memory to permit accessing of memory at a lower rate and higher duty cycle for information that is utilized at a higher rate and lower duty cycle. Various other features are also discussed herein.

Memory speed is an important consideration for design of digital systems; such as display systems, array processing systems, and pipeline systems. A configuration is discussed herein where system speed can be implemented to be significantly faster than implied by memory speed considerations. This configuration uses a combination of novel architectural features for outputting of relatively high bandwidth information with a relatively low bandwidth memory.

### Brief Description

A memory architecture in accordance with the present invention will now be discussed with reference to Figs 1P and 2M. Alternate configurations can be provided to implement the system of the present invention. However, this configuration is exemplary of the system of the present invention. Input device 115A generates input information under control of input clock 115G. Address generator 115B generates addresses for memory 115C under control of input clock 115G, such as for storing information from input device 115A in memory 115C or for accessing information from memory 115C under control of input device 115A. Memory 115C outputs information accessed with address generator 115B under control of input clock 115G. Buffer 115D receives information accessed from memory 115C for buffering therein under control of input clock 115G and generates information buffered therein under control of output clock 115F. Output device 115E, such as a display monitor, receives buffered information from buffer 115D under control of output clock 115F. This permits information to be accessed from memory 115C asynchronous with information to be output to output device 115E. Hence, buffer 115D can input information under control of input clock 115G and can output information under control of output clock 115F for resynchronizing of information flow, averaging of information rate, reorganizing of information into groups, and for other purposes.

In an alternate configuration, information from memory 115C can be output directly to output device 115E under control of input clock 115G, such as with input clock 115G and output clock 115F being the same clock and being connected theretogther.

A multi-dimensional address configuration is shown in Fig 2M. Address generator 115B generates an address word having a re-addressing portion 215E, a Y-scanout portion, 215G, and an X-scanout portion 215F. This arrangement has particular advantages because the X-scanout signal 215F and the Y-scanout signal 215G can be generated more rapidly than re-addressing signal 215E to access or to write into memory 115C.

Memory 115C is shown partitioned onto 2-boards 215B and 215D. RAMs on the 2-boards can be addressed with re-addressing logic 215E. The RAMs are shown organized in an X-Y array of rows and columns. The X-scanout signals are decoded into a plurality of row signals shown radiating horizontally right from the X-scanout line 115F. The Y-scanout signals are decoded into a plurality of column signals shown radiating vertically up from the Y-scanout line 115G. The decoded row and column line signals enabled 1-row and 1-column as a function of the X-scanout and Y-scanout address portions, respectively. Consequently, 1-RAM at the intersection of the row and column enable signals is enabled to output the information addressed with re-addressing signal 215E and all other RAMs are disabled from outputting the information addressed with re-addressing signal 215E.

Various address register configurations will now be discussed with reference to Figs 4A and 4B. Fig 4A shows a dual address register configuration having an X-address register and a Y-address register. This arrangement is particularly applicable to display systems having a 2-dimensional memory map and generating vectors for storing into an image memory or for reading out of an image memory. The X-address register and the Y-address register can be separately controlled to generate a 2-dimensional vector for accessing a pixel in the memory. Actually, the 2-address registers can be considered to be concatenated to form a single address parameter for memory accessing. However, two separate 2-dimensional address registers are a convenient way of visualizing a single dimensional memory configured into a 2-dimensional memory map. In a configuration discussed for the experimental system herein, the 6-most significant bits of each register are combined for an 11-bit re-addressing word and a 1-bit board select signal while the 3-least significant bits of each register are separately decoded to select one of 8-rows and one of 8-columns on each board in accordance with the X-scanout signal 215F and Y-scanout signal 215G discussed with reference to Fig 4A.

Fig 4B shows a single address register configuration, which can be implemented by concatenating the X-register and Y-register shown in Fig 4A or, alternately, may be conceptually defined as a single register, a quadruple register, or other configurations. Selected bits of this single address register may be used to control re-addressing and row and column select for the memory in accordance with Fig 2M. Many configurations of signal groupings

COPYA  
10/18/4

COPYA  
10/18/4

can be implemented; such as using signals XA0, XA1, XA4, XA6, YA3, YA4, and YA6 for re-addressing; signals XA3, XA7, YA5, and YA8 for X-scanout decoding; and signals YA0, YA1, YA2, XA2, XA5, XA8, and YA7 for Y-scanout decoding as an alternate to the above configuration discussed with reference to Fig 4A. ~~4A, 4B, 4C~~ 10/18/4

Memory addressing may be configured in a multi-dimensional form; such as 2-dimensional, 3-dimensional, or 4-dimensional form. For example, address generation may be performed with a plurality of different address generators; such as an X-address generator for one portion of a 2-dimensional address and a Y-address generator for the other portion of a 2-dimensional address, as discussed for a 2-dimensional memory map configuration herein. Also, addresses that are generated with a single address generator can be partitioned into multi-dimensional addresses; such as a 16-bit computer instruction address being partitioned into a W-dimension address for the most significant 4-bits, an X-dimension address for the next less significant 4-bits, a Y-dimension address for the next less significant 4-bits, and a Z-dimension address for the least significant 4-bits. Also, a multi-dimensional address, such as the X-dimensional address and Y-dimensional address for a display configuration, can be concatenated into a single address by combining the different portions thereof; such as combining the Y-address dimension as the most significant portion of the address word and the X-address dimension as the least significant portion of the address word.

In describing the novel architecture of the memory of the present invention, the word "dimension" and words relating thereto have been adapted to mean the different forms of addressing the RAMs. For example, the RAMs are addressed with a re-addressing portion of the address and a scanout portion of the address, which may be considered to be 2-dimensional addressing, and the scanout portion of the address are divided into row select signals to the G-bar pins and column select signals to the S-bar pins of the RAMs, which may be considered to be 2-dimensional scanout addressing. A memory having an address with the combination of a 1-dimensional re-addressing portion and a 2-dimensional scanout portion may be considered to be a 3-dimensional memory. This terminology is different from terminology associated with 2-spatial dimensions of an image, such as implemented in a memory map, and 2-spatial dimensions of an image, such as displayed on a monitor.

The memory arrangement discussed herein can be applied to a display system, as discussed in greater detail herein. Image pixels can be accessed in sequence for output to a display monitor. The memory can be configured in a 2-dimensional form, such as 1-dimension being the data block address and the other dimension being the pixel address within a block. Alternately, the address can be partitioned into a plurality of bytes from a least significant byte to a most significant byte and each byte can be used to address a different dimension of the memory. Image memory scanout can be implemented by scanning sequential addresses at a higher rate and by re-addressing the memory at a lower rate, such as with a gated clock. A gating signal can be

used to gate the memory access clock without gating the output clock, such as a DAC clock; permitting display operations to proceed under control of the non-gated output clock without being affected by gating of the memory clock. An output buffer memory can be used to temporarily store pixel information to reduce sensitivity of the display to gating of the memory clock. A buffer memory also permits accessing of the image memory at a relatively high duty cycle even though the information may be output to the display at a lower duty cycle, or at a relatively low portion of the time that the memory information is available, or by not utilizing the information immediately after the information becomes available from the image memory. A buffer memory also permits accessing of image memory substantially as fast as image memory can be accessed, reducing constraint from output speed considerations.

The memory arrangement discussed herein can also be applied to a correlator processor memory. Data can be accessed in sequence for execution by the correlator. The memory can be configured in a 2-dimensional form, such as 1-dimension being the data block address and the other dimension being the data address within a block. Alternately, the address can be partitioned into a plurality of bytes from a least significant byte to a most significant byte and each byte can be used to address a different dimension of the memory. Data scanout can be implemented by scanning sequential addresses at a higher rate and by re-addressing the memory at a lower rate, such as with a gated clock. A gating signal can be used to gate the memory access

clock without gating the output clock, permitting correlator operations to proceed under control of the non-gated output clock without being effected by gating of the memory clock. A buffer memory can be used to temporarily store correlation information to reduce sensitivity of the correlator to gating of the memory clock. A buffer memory also permits accessing of the correlator data at a relatively high duty cycle even though the information may be processed at a lower duty cycle, or at a relatively low portion of the time that the memory information is available, or by not utilizing the information immediately after the information becomes available. A buffer memory also permits accessing of correlator data substantially as fast as the data memory can be accessed, reducing constraints from output speed considerations.

The memory arrangement discussed herein can also be applied to an FFT processor memory. Data can be accessed in sequence for execution by the FFT processor. The memory can be configured in a 2-dimensional form, such as 1-dimension being the data block address and the other dimension being the data address within a block. Alternately, the address can be partitioned into a plurality of bytes from a least significant byte to a most significant byte and each byte can be used to address a different dimension of the memory. Data scanout can be implemented by scanning sequential addresses at a higher rate and by re-addressing the memory at a lower rate, such as with a gated clock. A gating signal can be used to gate the memory access clock without gating the output clock, permitting FFT operations to proceed under control of the non-gated output clock without

being effected by gating of the memory clock. A buffer memory can be used to temporarily store FFT information to reduce sensitivity of the FFT processor to gating of the memory clock. A buffer memory also permits accessing of the FFT data at a relatively high duty cycle even though the information may be processed at a lower duty cycle, or at a relatively low portion of the time that the memory information is available, or by not utilizing the information immediately after the information becomes available. A buffer memory also permits accessing of FFT data substantially as fast as the data memory can be accessed, reducing constraints from output speed considerations.

The memory arrangement discussed herein can also be applied to general purpose computer memory. Instructions can be accessed in sequence for execution by the computer arithmetic and control logic. The memory can be configured in a 2-dimensional form, such as 1-dimension being the instruction or data block address and the other dimension being the instruction or data address within a block. Alternately, the address can be partitioned into a plurality of bytes from a least significant byte to a most significant byte and each byte can be used to address a different dimension of the memory. Instruction and data scanout can be implemented by scanning sequential addresses at a higher rate and by re-addressing the memory at a lower rate, such as with a gated clock. A gating signal can be used to gate the memory access clock without gating the output clock, permitting processing to proceed under control of the non-gated output clock without being effected by gating of the memory clock. A buffer

memory can be used to temporarily store computer instructions and data to reduce sensitivity of the computer to gating of the memory clock. A buffer memory also permits accessing of the computer instructions and data at a relatively high duty cycle even though the information may be processed at a lower duty cycle, or at a relatively low portion of the time that the memory information is available, or by not utilizing the information immediately after the information becomes available. A buffer memory also permits accessing of computer instructions and data substantially as fast as the memory can be accessed, reducing constraints from output speed considerations.

Memory architectural features pertaining to high speed scanout in conjunction with re-addressing can provide speed enhancement, such as a 3-fold improvement in speed. These features are particularly pertinent to RAMs having multiple tristate control signals; such as the Mitsubishi Electric M58725P RAMs. RAMs having a single tristate control signal can also be used with this configuration, but may involve additional decoder logic to decode scanout address signals, such as with linear select architecture consistent with a single tristate control signal.

The memory architecture of the present invention may be discussed in the context of a display application for purposes of illustration. However, this memory architecture is applicable to computer main memories, buffer memories, signal processing memories, and other memory applications in addition to display memories.

### Re-Addressing And Scanout Memory Architecture

Multiple dimension image memory architecture, as previously discussed, involves simultaneous accessing of multiple pixels, such as in a 2-dimensional X/Y array, to increase effective memory speed. Such a configuration is appropriate for a 2-dimensional horizontal and vertical scanout for refreshing a display monitor and is also appropriate for other applications; such as general purpose computers and special purpose processors. Such a memory architecture may need buffer registers for temporary storage of accessed information, such as for temporary storage of accessed pixels so that a new memory access cycle may be initiated while the previously accessed pixels are being output to refresh the display monitor. An alternate configuration is discussed herein where a block of pixels is simultaneously accessed and is scanned-out without the need for buffer registers or overlapping memory accesses. This configuration can involve a multiple access period, where stored information is scanned out from an accessed block at high rate (shorter period) and a new block of stored information is accessed at a lower rate (lower period). A buffer memory, such as a FIFO or a double buffer memory, can be used to equalize these rate and period differences.

A novel memory architecture will now be discussed which enhances memory speed and economy. This architecture can be characterized as a multi-dimensional memory architecture that is divided into 2-address portions, a high speed address portion and a slow speed address portion. Another characteristic is a combination scanout and re-addressing architecture. Another

characterization is use of tristate memory control logic to reduce the need for buffer registers and multiplexing logic. This can be accomplished by taking advantage of certain features of RAMs.

Conventional RAMs have a plurality of input address lines for addressing stored information, tristate output data direction control logic for selecting data input for writing and data output for reading, and tristate chip select logic for gating output information onto a bus. Use of these circuit features in a novel form implemented in the memory architecture described herein provides important advantages. For example, tristate data input and output control logic can be used in conjunction with tristate chip select logic to provide a high speed 2-dimensional scanout for rapid accessing of RAMs. The 2-dimensional scanout arrangement reduces auxiliary decoding and selection logic, reduces output buffer logic, is compatible with 2-dimensional memory map architectures, and facilitates relatively high speed operation with relatively low speed RAMs.

Higher speed scanout can be used in conjunction with slower speed addressing of the RAMs to provide an average access rate that is significantly higher than the addressing rate. For example, the Mitsubishi Electric M58725P RAMs have a 200-ns address period and a 100-ns scanout period. Assuming that a system will scanout 4-parameters before re-addressing is necessary and assuming that re-addressing is implemented with 3-scanout clock periods, 5-parameters can be accessed in 7-clock periods; 4-parameters times 1-clock period per parameter plus 1-

parameter times 3-clock periods per parameter; in comparison to conventional re-addressing, where 5-parameters can be accessed in 15-clock periods (5-parameters times 3-clock periods/parameter). This scanout and re-addressing example yields an average of 1.4-clock periods per pixel for the scanout and re-addressing configuration compared to 3-clock periods per pixel for the re-addressing configuration, yielding an improvement of about 2-times in speed for this example.

RAMs are conventionally addressed with a number of address lines, such as 11-address lines for a 2,048 word RAM. Address signals typically propagate through the memory array and consequently can have relatively long propagation delays. RAMs conventionally have tristate enable signals to permit bussing of output signals and to select data input for storing and data output for accessing of data. The tristate enable signals can be used to gate the RAM outputs and consequently can have relatively short propagation delays.

The present multi-dimensional memory configuration uses less frequent accessing of data with the slower address signals (re-addressing) and uses more frequent accessing of data with the faster scanout control signals. Therefore, the average propagation delay is reduced, being a weighted average of several shorter scanout propagation delays and a single longer address propagation delay.

Speed is enhanced by changing the clock period to be a function of the addressing operation, such as a longer clock period for re-addressing and a shorter clock period for scanout. A buffer memory; such as a FIFO, double buffer, cache, or

scratchpad memory; can be used to buffer output information from the memory for providing a constant memory output clock period in response to the variable memory input clock period.

A specific example will now be provided to illustrate use of relatively longer propagation delay address signals to select a single block of 64-pixels and using relatively shorter propagation delay tristate control signals to select a pixel from the selected block. Each address generator generates a concatenated address having a 3-bit tristate control signal portion and a 6-bit address signal portion. The 3-bit tristate control signal address can be implemented with the least significant bits (LSBs) of the address word and the 6-address bits can be implemented with the most significant bits (MSBs) of the address word. The address generators can be implemented to update the address, where the LSBs can be updated more frequently than the MSBs and where the addresses can scan through a block of pixels as the LSBs are updated and change to a different block of pixels when the MSBs are updated. Updating of the MSBs can be detected with an overflow from the LSBs to the MSBs in the address generator. Therefore, the LSBs can be updated relatively rapidly to scanout with the faster tristate control signals through the pixels within a block and the MSBs can be updated relatively slowly to change the selected block.

For purposes of illustration, an experimental configuration with an image memory having 262,144-pixels arranged in a 512-by-512 pixel memory map was implemented. Also for simplicity, Mitsubishi 58725P RAMs, Texas Instruments 7400 series TTL logic,

and Intel logic is used. The Mitsubishi RAMs have 2K-words by 8-bits per word. Therefore, 128-RAMs are used to provide 262,144-pixels. For efficiency of implementation, the RAMs are arranged in two 2-dimensional boards each having a binary quantity (i.e., 8) RAMs for each dimension. Consequently, the 128-RAMs are arranged on 2-boards each having 64-RAMs, arranged in an 8-by-8 block of RAMs per board.

Each RAM has an 11-bit address for accessing one of 2048-words. Another address bit is used in this configuration for selecting one of the two RAM boards. For convenience of discussion, the 11-bit address and the 1-bit board select signals are organized in a 6-bit X-address dimension and a 6-bit Y-address dimension to select one block of 64-pixels out of 4096 blocks of 64-pixels. This arrangement is shown in the memory diagrams and tables included herewith. The memory map contains a 64-by-64 array of blocks for a total of 4096 blocks. The 12-address bits are organized into a 6-bit Y-axis address and a 6-bit X-axis address for a 64-by-64 array of blocks. The 6-bit X-axis address is divided into a 5-bit X-axis address to each RAM and a 6th X-axis address bit to select one of the two 64-RAM boards. Use of the X-address bit as a board select bit causes the 64-by-64 array of RAMs to have alternate X-dimensional columns to be selected from different boards. Alternately, use of the most significant X-address bit for the board select bit would cause the 64-by-64 array of RAMs to have all of the X-dimensional RAMs in one board adjacent to each other and all of the X-dimensional RAMs in the other board adjacent to each other.

Each of the 4096-blocks of pixels can be configured in an 8-by-8 array of 64-pixels. One of the 8-by-8 arrays of 64-pixels is shown in the image memory diagrams and tables included herewith. The 8-by-8 array can be addressed with a 2-dimensional 3-bit by 3-bit address organized in a 3-bit X-address and a 3-bit Y-address format. Each of the two 3-bit address portions can be decoded into 8-address lines, yielding an 8-by-8 array of address lines. If one of the first group of 8-address lines is excited to select a row of pixels and the second group of 8-address lines is excited to select a column of pixels, then the one pixel at the intersection of the row address line and column address line is selected out of the 64-pixels per block.

The memory scanout and re-addressing architecture can be implemented for cutting across image memory lines to generate a vector drawn at an angle to the raster lines and can be efficiently used in a raster scan image memory. For example, raster scan outputs proceed on a line-by-line basis; consistent with the line organization of the image memory. In such an arrangement, a single tri-state control signal can be used for scanout as the line progresses, with re-addressing being performed at block boundaries. In this configuration, a block may be a 1-dimensional line of pixels; in contrast to the above described 2-dimensional array of pixels per block. Block traversing in a 1-dimensional memory system is more nearly constant than in a multi-dimensional memory system. For example, a linear block along a scanline can have all pixels in the block accessed frame-after-frame independent of vector considerations. This may be different from a system having a 2-dimensional

scanout block with vectors because a 2-dimensional scanout block may have different numbers of pixels traversed within a block as a function of the vector parameters. For example, the number of pixels traversed may be a function of the pixel entry point to the block and a function of the scanout vector angle through the block, where the pixel entry point may be a function of vector position and the scanout angle may be a function of vector slope. The maximum, typical, average, and minimum number of pixels scanned in a block can be different for different configurations. For example, in a raster scan arrangement; all pixels in a block may be scanned for each traverse of the linear block. Therefore, in a 1-dimensional block configuration; the maximum, typical, average, and minimum number of pixels scanned in a block may be the same; which is the total number of pixels per block. In a 2-dimensional block configuration having vector directions, the maximum number of pixels may be the number of pixels along the diagonal of the block; the minimum number of pixels may be a single pixel, such as a scan clipping the corner of a block; and the typical number and average number may be between the maximum and minimum, such as determined by typical linear block raster scanout relationships. Consequently, in a linear block raster scanout configuration, greater average scanout rates may be obtained due to the scanout of more pixels per block.

A buffer memory configuration can have the form of a multiple buffer, such as a double buffer. Alternately, the memory configuration discussed herein having gated clocks and a

plurality of clock periods can be implemented with a smaller buffer memory to average a plurality of clock periods. For example, a 16-word FIFO can be used for loading at an input word rate under control of the gated clock pulse having a plurality of clock periods and for unloading at a clock rate consistent with the output word rate. The FIFO may be implemented with the S/N 74LS222, S/N 74LS224, or S/N 74S225 circuits which provide 16-words having 4-bits or 5-bits. These FIFOs can be configured in parallel to provide the 8-bits per word, as shown for the experimental configuration, or can provide other word sizes.

### Memory Enhancement

The output of memory in the experimental configuration can be implemented to propagate down a datapipe to a buffer memory without communication with the front end control logic and address generators. Therefore, it can be considered to be implemented in a pipeline form. This pipeline permits introducing clock skew, where the clock to the output data pipe can be skewed ahead of the clock to the input at the data pipe. This permits greater speed in view of propagation delay considerations. For example, in the experimental configuration, the output registers are clocked with the 180-degree-phase clock and, consequently, provide a 1.5-clock period for propagation delay. One design consideration for this configuration is that propagation delay through the memory is less than 1/2-clock period or the newer high speed information from the next clock may be clocked into the datapipe with the 1.5-clock period clock.

### Memory Map Display Architecture

An architecture of one form of memory map for a display will now be discussed. This memory map stores an image as a 2-dimensional array of pixels. In a monochromatic configuration, each pixel contains 1-intensity parameter. In a color configuration, each pixel contains 3-intensity parameters; red, blue, and green intensity parameters. Additional information can be contained in a pixel word; such as other parameters, flags, and control information. All information for a particular pixel can be packed together in a pixel word. Accessing of a pixel word can be implemented to access all information pertaining to the particular pixel word with one access or, alternately, can access portions of the pixel word for each of multiple accesses per pixel word. For simplicity of discussion herein, operations on a pixel word may be discussed as operations on a pixel.

Memory map configurations having 1-dimensional and multi-dimensional architectures have been discussed. Multi-dimensional architectures provide enhanced performance and flexibility, such as by accessing multiple pixels simultaneously.

Multi-dimensional architecture provides important advantages. It provides high speed, because of the addressing of 64-pixels in parallel and because the tristate select signals have shorter propagation delays than the address signals. It provides flexibility, because the tristate select signals can traverse the 64-pixel block at any vector angle and through any continuous sequence of pixels. It provides circuit efficiency because much of the address decode and tristate logic is implemented on the memory chips, because a multi-dimensional

addressing arrangement is more efficient than a 1-dimensional addressing arrangement, and because the tristate logic reduces the need for output registers.

A configuration for simultaneous accessing of an 8-by-8 2-dimensional array of 64-pixels is shown in Figs 6E to 6N. A pixel address selects 64-pixels at a time out of the total array of pixels, such as out of 262,144-pixels in a 512-pixel by 512-pixel array. A subset of the 64-pixel block is then addressed with tristate enable signals, such as chip select and output select signals. Tristate select signals can be used to scan through a 64-pixel block to select a sequence of pixels therefrom.

The experimental configuration has been constructed having a 512-pixel by 512-pixel memory map. For convenience of experimentation, static 16K-RAM chips are used for memory map implementation. Typical circuits are the TMS-4016 RAM from Texas Instruments Inc. and the M58725P static RAM from Mitsubishi Electric. These circuits are configured in the form of a 2K-word by 8-bit static RAM having an 11-bit address, a tristate chip select, and a tristate output enable. Conventionally, the chip select and output enable are used to provide output bussing and to reduce the need for an output register. In the present configuration, the chip select and output enable signals are used to provide 2-additional dimensions of memory addressing. For example, the 11-address lines are used to select a block of 64-pixels out of 262,144-pixels; the chip select signal is used to select a column of 8-pixels out of the selected block of 64-

pixels; and the output enable signal is used to select a row of 8-pixels out of the selected block of 64-pixels. The selection of a column of 8-pixels and a row of 8-pixels with the chip select signal and the output enable signal selects a single pixel at the intersection of that column and row from the selected block of 64-pixels. Consequently, a 3-dimensional architecture having address selection of a 64-pixel block, chip selection of an 8-pixel column in that block, and output enable selection of an 8-pixel row in that block uniquely selects a single pixel out of 262,144-pixels.

Memory chips have particular characteristics that can be adapted to memory architectures which are particularly appropriate for the systems discussed herein. For example, memory circuits conventionally have address lines and tristate select lines. The address lines are typically used to select a pixel per chip and the tristate select lines are typically used to disconnect undesired chips from the output bus and to reverse data direction for read and write operations. However, use of the address lines to select a block of pixels and use of the tristate select lines to scan through the block of pixels provides particular advantages. The access time from the address select is significantly greater than the access time from the chip select or the output enable select. Therefore, accessing with the chip select and output enable signals can proceed at a significantly faster rate than accessing with the address select.

The address select lines can be excited with the most significant bits (MSBs) of the X-address and Y-address generated with the address generators. The chip select and output enable

signals can be excited with the decoded least significant bits (LSBs) of the X-address and Y-address generated with the address generators. For a 512-pixel by 512-pixel memory map having blocks of 64-pixels, the 6-MSBs of the Y-address and the 6-MSBs of the X-address can be combined into a 12-bit address to select one of 4096 blocks of 64-pixels. The 3-LSBs of the Y-address and the 3-LSBs of the X-address can be used to select one of 8-rows and one of 8 columns, respectively. As the address generation proceeds within a block of pixels, the address proceeds along a line at the appropriate vector angle through the block as the 3-LSBs of the X-address and the 3-LSBs of the Y-address are updated. When the address update progresses into the MSBs of either the X-address or Y-address, such as with an overflow; a new block of pixels is accessed and the address generation then proceeds within this new block of pixels along a line at the appropriate angle through the block.

The time period for memory accessing of a new block of pixels can be implemented to be longer than the time period for scanning pixels within a previously accessed block of pixels. This can be provided by scanning the pixels within a block at a higher rate and then accessing a new block at a lower rate. This can be implemented by using a higher clock rate to scan pixels within a block, to detect an overflow condition in the X-address and Y-address generators from the LSBs to the MSBs as being indicative of the need to access a new block of pixels, and to switch over to a lower clock rate for accessing of the new block of pixels.

The arrangement discussed with reference to Figs 6E to 6N illustrates 64-RAM chips in an 8-by-8 array of chips. Two of these 8x8 arrays of chips are used for a 512-pixel by 512-pixel memory map that is implemented in 64-pixel blocks with 2K-by-8 static RAMs. Each 2K-block array of pixels is selected with 11-bits of the 12-bit address. The particular one of the two 64-chip boards is selected with the remaining bit of the 12-bit address. This is shown with the 5-bits from the X-address and the 6-bits from the Y-address being bussed to all 128-chips of both boards and with 1-bit of the X-address being used to select one of the 2-boards in the uncomplemented state and the other of the 2-boards in the complemented state. The row select and the column select are each implemented by decoding 3-bits with a decoder to generate one of 8-signals to select one of 8-rows and one of 8-columns, respectively. The 11-address lines are bussed to the address input lines of each RAM chip. The 8-data lines are bussed from the data output lines of each RAM chip. Each column select line is bussed to all 8-chips in the related column for each of the two blocks. Each row select line is bussed to all 8-chips in the related row for each of the 2-boards. Consequently, the 11-address lines select 2-boards of 64-chips each, the twelfth address line selects one of 2-boards, and the 6 "scarnout" lines select one of 64-pixels per board.

### Image Memory

An image memory for a display in accordance with the present invention can take various forms; such as being implemented with static RAMs, dynamic RAMs, CCDs, ROMs, and other memory devices. The memory architecture can be a random access, sequential access, block access, or other form of architecture. The image memory can be implemented in an unbuffered form, or in a buffered form; such as with a double buffer, in conjunction with various line buffers, and in conjunction with frame buffers. These various alternatives can be adapted to operate with the present invention based upon the teachings herein showing a detailed design of a RAM image memory using static RAMs and accessed in a block oriented scanout arrangement. This configuration will now be discussed in detail with reference to Figs 6E to 6N.

Address generators for use with the memory arrangement shown in Figs 6E to 6N are discussed with reference to Figs 6O to 6R. The address generators can generate sequential addresses at the appropriate vector angle through image memory. Multiple RAM chips, in this example all RAM chips, are addressed with the more significant bits of the same address word for simultaneously accessing the corresponding word in each of the multiple RAM chips. The less significant bits of the address word are used to select which of the chips is to be enabled for outputting onto the output bus. The chip enable control is a higher speed control and hence permits higher speed memory operations when scanning out within a memory block and the chip address control is a lower speed control and hence involves lower speed memory operations when re-addressing. Therefore, two types of addressing will be

described with reference to Figs 6E to 6N, which are re-addressing with the more significant address bits and scanout with the less significant address bits.

Re-addressing is performed with fanout buffers U19A for the Y-address bits and U19D for the X-address bits. These buffers generate the drive current necessary to fanout to a large number of RAM chips. In this configuration, 64-RAM chips are grouped on each of two image memory boards with each board having replicated buffers to facilitate increased speed and modularity. The buffer outputs are applied to the address inputs of the RAM chips. During scanout, the addresses are maintained constant. During re-addressing, the addresses are changed.

Scanout is performed with decoders U19B, U19C, and U19E. The less significant address bits are applied to these decoders and decoded into X-address and Y-address signals. The X-address signals select rows of RAM chips and the Y-address signals select columns of RAM chips in a 2-dimensional configuration on each board. Replicating memory address logic on each board facilitates increased speed and modularity. Each row and each column is composed of 8-RAM chips for an 8-by-8 array of RAM chips per board.

The Y-axis decoder U19B is addressed with the less significant Y-address bits YA0, YA1, and YA2 for decoding of column signals. The decoded column signals are applied to the RAM chip select pin, pin 18, to select the column of RAM chips and are applied to the Intel 8216 bus interface chips associated with that column for outputting to the memory output bus.

Selection of one of the 2-boards is provided with the fourth from the least significant X-address bit XA3 applied to U19B-6 and to the Intel 8216 chip select logic. XA3-bar is used to select memory board-1, XA3 is used to select memory board-2. Therefore, as the scanout proceeds in the X-direction, the same block on alternate boards are selected without re-addressing; effectively providing a 16-column by 8-row aspect ratio of RAM chips. Enabling of U19B with XA3 or XA3-bar to pin-6 is an optional control; where selection of one of two memory boards is performed with the Intel 8216 bus interface logic, as described below. Gating the column addresses U19B-6 with the XA3 and XA3-bar signals reduces memory power consumption.

The X-axis decoders U19C and U19E are addressed with the least significant X-address bits XA0, XA1, and XA2 for decoding of row signals. Decoder U19C is used for read operations, where the decoded row signals are applied to the RAM data enable pin, pin 20, to select the row of RAM chips for read operations. The RAM data enable control, pin 20, is conventionally used for selecting data direction during read and write operations. However, in this configuration; it is also used to facilitate 2-dimensional scanout capability. Decoder U19E is used for write operations, where the decoded row signals are applied to the write control pin, pin 21, to select the row of RAM chips for write operations. The DIEN-bar signal is generated from the computer run/load-bar signal DOA6; enabling U19C-6 for read operations during the run mode when DIEN-bar is 1-set, disabling U19C-6 for read operations during the load mode when DIEN-bar is 0-set, enabling U19E-5 for write operations during the load mode

when DIEN-bar is 0-set, and disabling U19E-5 for write operations during the run mode when DIEN-bar is 1-set. Therefore, during the run mode, U19C is enabled for reading image memory and U19E is disabled for preventing writing into image memory. Also, during the load mode, U19C is disabled to prevent reading of image memory and U19E is enabled for permitting writing into image memory. Enabling of U19E enables the write pulse W-bar input to U19E-4 to be steered to the appropriate row of RAM chips for writing into the RAM chip that is enabled with the chip select column signal to pin 18.

The RAM chip data lines carry the output byte from the selected RAM chip during read operations and carry the input byte to the selected RAM chip during write operations. A shared bi-directional bus structure is used for bi-directional communication with the RAM chips. Intel 8216 bus interface circuits are used for bi-directional communication between a read bus and a write bus and the RAM chip. Each Intel 8216 can accommodate 4-lines, where Intel 8216 chips are used in pairs; U17A and U18A, U17B and U18B, U17C and U18C, and U17D and U18D; to accommodate the 8-lines of a RAM data byte. As can be seen in the memory schematics (Figs 6E to 6N); on the system bus side, the 8-input unidirectional lines for each pair of Intel 8216s are connected to different lines on the system write bus and the 8-output unidirectional lines for each pair of Intel 8216s are connected to different lines on the system read bus. On the memory side, the bi-directional input and output buffers of the 8216s are internally connected together to provide 8-bit

directional lines connecting to the 8-lines for each RAM chip associated with the particular pair of Intel 8216s. The design connects the data buses for a pair of columns of RAM chips to a single pair of Intel 8216s. This facilitates a tradeoff of the number of Intel 8216 chips used and the speed of operation.

A pair of NAND-gates U17E and U18E are used to OR the two column select signals associated with the pair of Intel 8216s and to AND the board select signal XA3 or XA3-bar for selection of the pair of Intel 8216s as a function of the selected board and the selected column pair of RAMs on that board. The selected pair of Intel 8216s connect the selected RAM to the input bus or output bus under control of the above described signals.

The run/load-bar signal applied as the DIEN-bar signal on pin 15 of the Intel 8216s selects the direction of data communication. If the DIEN-bar signal is 1-set, indicative of the run mode of operation; the 1-set signal applied to pin-15 of an Intel 8216 commands data output from the RAM data bus to the memory output data bus for reading of RAM. If the DIEN-bar signal is 0-set, indicative of the load mode of operation; the 0-set signal applied to pin-15 of an Intel 8216 commands data input to the RAM data bus from the memory input data bus for writing into RAM.

During read operations, all RAM chips are addressed with the same address signals and one of the RAM chips is selected with a combination of a column select signal and a row select signal. The selected RAM chip will have its output data lines enabled to be applied to the output data bus through the Intel 8216 bus interface chips. The column select signal also selects the Intel

8216 bus interface chips associated with selected column for applying the column-related RAM bus to the data bus for reading.

During write operations, all RAM chips are addressed with the same address signals and one of the RAM chips is selected with a combination of a column select signal and a row-selected write pulse for writing the information from the data lines into the selected RAM chip. The column select signal also selects the Intel 8216 bus interface chips associated with the selected column for applying the input data from the data bus to the RAM chips for writing.

### Improved IC Memory Chip

An improved IC memory chip, can be implemented in accordance with the teachings of the present system and can provide important advantages over conventional memory chips. This improved memory chip can have multiple tristate control chip select inputs, similar to the 2-dimensional arrangement of the chip select and data enable signals. Also, each memory chip can have an output register to latch the accessed data, where the output register has an output tristate select with multi-dimensional selection. The data can be latched for scanout and new data can be accessed with a changing address. The data can be strobed into the output register before re-addressing the RAM, such as with a data hold strobe.

Multiple dimensions of tristate output select, can be implemented, exemplified by the 2D tristate control of the system disclosed herein. 2D, 4D, and other multi-dimensional tristate controls can provide further advantages in decoding and scanning-out from image memory.

### Memory Logical Design

The memory implemented for the experimental configuration is implemented in a multiple board arrangement, where each board contains 64-RAMs organized in a logical 8-RAM column by 8-RAM row 2-dimensional array. All RAMs receive the same address. All 8-RAMs in an 8-RAM row receive the same X-select signal, which is different from the X-select signal for all other rows. All 8-RAMs in an 8-RAM column receive the same Y-select signal, which is different from the Y-select signal for all other columns. The input data and output data signals are bussed together for groups of 16-RAMs and interfaced with Intel 8216s for connecting to the system databus.

The regular array of RAMs lends itself to a tabular wire list type of documentation. MEMORY TABLE-A to MEMORY TABLE-D list the interconnections for the 64-RAM array on a board. The RAMs are organized in a physical 4-row by 16-column array comprising row-A to row-D and column-1 to column-8. Each RAM is identified by the physical row and column designation; where RAM U1A is the RAM that occupies the row-A and column-1 position, RAM U6C is the RAM that occupies the row-C and column-6 position, and the other RAMs occupy the other positions in row-A to row-D and column-1 to column-16.

MEMORY TABLE-A lists the connections for pin-1 to pin-12 of the first group of 32-RAMs. MEMORY TABLE-B lists the connections for pin-1 to pin-12 of the second group of 32-RAMs. MEMORY TABLE-C lists the connections for pin-13 to pin-24 of the first group of 32-RAMs. MEMORY TABLE-D lists the connections for pin-13 to pin-24 of the second group of 32-RAMs.

The address connections are the same for all RAMs; where pins 1 to 8, 19, 22, and 23 are connected to an 11-wire address bus; where each bus wire connects the same pin on each RAM. The vertical scanout pin, pin 18, for all 8-RAMs in a logical column are connected together and are connected to the vertical scanout signal from U19B corresponding to the particular logical column. The horizontal scanout pin, pin 20, for all 8-RAMs in a logical row are connected together and are connected to the horizontal scanout signal from U19C corresponding to the particular logical row. The horizontal write pin, pin 21, for all 8-RAMs in a logical row are connected together and are connected to the horizontal write signal from U19E corresponding to the particular logical row.

The databus connections are the same for all RAMs in a double logical column or single physical column array; where pins 9 to 11 and 13 to 17 are connected to an 8-wire data bus connecting all 16-RAMs in the double logical column or single physical column group. A pair of Intel 8216s connect each 16-RAM databus to the system databus with bi-directional read and write signal paths. Four pairs of Intel 8216s bi-directionally connect all 64-RAMs to the system databus.

### Other Memory Configurations

F Various configurations of the memory of the present invention have been above to illustrate how the various features and devices of the memory of the present invention can be used to implement a system. These configurations are illustrative of a large number of other configurations that can be implemented from the teachings herein.

The memory configuration of the present invention has been discussed relative to implementing a 2D memory map for an image processing system and has briefly been discussed for other applications. It is herein intended that the memory architecture of the present invention be usable with other types of display systems and with other systems, such as computer systems and signal processing systems, that are not display systems.

The memory configuration of the present invention has been discussed in memory map form with an address derived from an X-axis address component and a Y-axis address component. Alternately, other addressing configurations can be implemented; such as a single address component for what may be considered to be a 1D memory, a 3-address component for what may be considered to be a 3D memory map, and other memory addressing configurations.

The memory configuration of the present invention has been discussed with reference to an integrated circuit RAM of the Mitsubishi M58725P-type. However, the teachings of the present invention are also appropriate for other integrated circuit RAMs and are also appropriate for integrated circuit ROMs and other memory technologies.

The memory configuration of the present invention has been discussed for a RAM component having 2-tristate control signals for controlling the tristate input and output of the RAM. Alternately, other numbers of tristate control signals can be accommodated; such as 1-tristate control signal, 3-tristate control signals, 5-tristate control signals, and other quantities of tristate control signals. The architecture for 2-tristate control signals permits implementation of what may be termed a 2-dimensional scanout arrangement having X-scanout control signals and Y-scanout control signals. Alternately, for a configuration having RAMs with 1-tristate control signal, a memory architecture that may be termed a 1-dimensional scanout arrangement can be implemented having 1-scanout signal to each RAM. The 1-scanout signal may be a single dimensional decode of the scanout portion of the address; such as 6-scanout bits being decoded to 64-RAM control signals with a different one of the 64-control signals going to each RAM tristate control signal input. Alternately, for a configuration having RAMs with more than 2-tristate control signals; such as 3-tristate control signals; a memory architecture that may be termed a multi-dimensional scanout arrangement; such as a 3D scanout arrangement; can be implemented with multiple scanout signals to each RAM; such as 3-scanout signals to each RAM. For example, the scanout portion of the address word can be divided into 3-groups of scanout signals, similar to the 2-groups of scanout signals for the arrangement discussed with reference to Figs 6E to 6N, and 1-signal from each of the 3-groups of scanout signals can be applied to a different

one of the 3-tristate control inputs to the RAM for what may be considered to be a 3D-scanout control arrangement.

The memory configuration of the present invention has been discussed for an arrangement that applies the same re-addressing portion of the address word to all RAMs. Alternately, the re-addressing portion of the address word can be partitioned to different RAMs; such as with decoding of a portion of the address and selecting blocks of RAMs with the decoded signals, such as to the chip select pin of the RAMs.

The multi-dimensional memory addressing arrangement discussed herein has been illustrated with reference to RAMs having 2-tristate control pins. Such multi-dimensional addressing can be implemented with a memory having a single tristate control pin, as discussed above, with digital logic to convert the single tristate control pin to a multi-dimensional scanout addressing arrangement. For example, the single tristate control pin, if implemented in complement logic form for selecting with a complement signal, can be accessed with a 2-dimensional scanout arrangement by NANDing the 2-scanout address signals, such as the row select signal and the column select signal, with a NAND-gate to control the single tristate pin of the RAM. Similarly, multiple dimensional scanout control signals can be combined with logic external to the RAM to adapt the external scanout control signals to the particular capabilities of the RAM. For example, a 6-dimensional scanout arrangement can be adapted for a 3-dimensional tristate controlled RAM by combining the 6-dimensional scanout signals into pairs processed with two input NAND-gates to control the 3-tristate pins.

Alternately, this 6-dimensional scanout arrangement can be adapted for a 2-dimensional tristate controlled RAM by combining the 6-dimensional scanout signals into groups of 3-signals processed with 3-input NAND-gates to control the 2-tristate pins.

The memory configuration of the present invention has been discussed for an arrangement that pre-buses the data lines of 16-RAMs into a pre-databus and then further buses the 16-RAM pre-bused signals together onto a system databus. Other partitioning of bused data signals can be provided. For example, all data signals can be bused together onto the system databus without the intervening pre-busing of the 16-RAM data outputs. Alternately, other combinations of RAMs than 16-RAMs can have the data lines pre-bused, such as pre-busing of the data lines of 8-RAMs together.

BUFFER MEMORY

### General

A buffer memory can provide important capabilities, such as discussed herein. For example, in a display application; a buffer memory permits the output pixel rate to be independent of the input pixel rate, such as for inputting at a rate consistent with the image processor and image memory characteristics and outputting at a rate consistent with CRT monitor characteristics. Further, it permits the CRT monitor to be relatively independent of discontinuities in the image processor, such as permitting the image processor to change the image processing clock period, without causing an undesirable transient on the CRT monitor. Still further, it permits the image processor to operate for the full blanked and unblanked line periods even though the CRT monitor may only be displaying information during the unblanked period; such as by loading the buffer memory during both, the blanked and unblanked periods, and unloading the buffer memory only during the unblanked periods. Yet further, it permits a multi-pixel kernel to be available to a spatial filter without requiring the image processor to make multiple redundant memory accesses of kernel pixels.

A buffer memory can be implemented <sup>"</sup>inbetween an image memory and a display monitor to provide various capabilities. For example, in a spatial filtering arrangement; a buffer memory can provide a 2-dimensional kernel of 9-pixels for each center pixel without the need to provide 9-pixel accesses from the image memory for each center pixel. Also, a buffer memory can provide greater pixel resolution by permitting accessing of a slower image memory during the full line period, blanked and unblanked,

at a lower pixel rate and accessing the faster buffer memory during the unblanked line period at a higher pixel rate. Further, a buffer memory permits use of a slower speed image memory because of the ability to access the image memory during the full line period, blanked and unblanked, and the ability to use the block partitioning arrangement of image memory with a high speed scanout implementation. Still further, a buffer memory permits the image memory to be accessed at a rate different from the display pixel rate and permits the image memory to be accessed at a non-constant rate and an asynchronous rate relative to the display monitor refresh rate; such as for equalizing different rate pixel output periods from the image memory caused by a fast scanout rate and a slower block transition rate.

The memory architecture discussed herein has many advantages. For example, a memory configuration is provided for high resolution pixel rates to a display monitor using slow RAMs for image memory. Also, a configuration is provided for generating a kernel of 9-pixels to be available in parallel for image processing using an image memory that is accessed at a lower rate. Also, a configuration is provided for buffering of accessed information that is to be used a plurality of times as an alternate to multiple accesses for information that is to be used a plurality of times.

A first-in first-out (FIFO) memory can be used as a buffer memory. This permits substantially simultaneous loading and unloading of the buffer memory and permits time sharing of a

single buffer memory between loading and unloading operations. Contention between loading and unloading operations can reduce effective speed. For example, with unloading requirements determined by the display monitor pixel rate and with image quality effected by changes in the output pixel period, the memory speed can be kept relatively high and the output pixel period can be kept constant. This can be accomplished by resolving contention in favor of the output accessing of the FIFO, effectively reducing the input rate due to contention.

A multiple buffer arrangement, such as a double buffer arrangement, can be used for a buffer memory. A multiple buffer is configured with redundant memory circuits, where one buffer is being loaded while another buffer is being unloaded, consequently reducing contention therebetween. Therefore, loading and unloading of the buffer memory can proceed at the write and read rates, respectively, for the memory circuits without reducing the rates as a result of contention. An advantage over a FIFO can be obtained with a multiple buffer because a FIFO involves both read and write operations in the same memory at substantially the same time while a multiple buffer arrangement permits either read operations for outputting from the buffer without write operations or permits write operations for inputting to a buffer without read operations; thereby permitting slower RAM circuits and greater pixel rates.

A double buffer memory can be implemented with the required buffer being replicated with double the amount of RAM circuitry. For example, a 1-line buffer configuration would be implemented with a double 1-line buffer for storing of an image in a first

line-buffer while accessing a 1-line image from a second line-buffer. Similarly, a 3-line buffer configuration would be implemented with a double 3-line buffer for storing of a 3-line image in a first 3-line buffer while accessing a 3-line image from a second 3-line buffer. An alternate configuration provides a multiple buffer, such as a double buffer, without the need to double all of the buffer memory capability. For example, a system having a 3-line buffer for convenient generation of a 9-pixel kernel can be implemented with a 3-line buffer for the output buffer and a 1-line buffer for the input buffer and having precession or commutation of line buffers between input and output operations. This can reduce memory requirements, such as reducing requirements from a 6-line full double buffer to a 4-line improved buffer; yielding a reduction in memory circuit requirements by one-third.

Memory architecture pertaining to a buffer memory provides further speed enhancement in addition to the scanout and re-addressing features discussed above. For example, in a display application image memory can be accessed during the full line period, blanked and unblanked portions, and pixels can be output to the monitor during the unblanked portion and not during the blanked portion. This provides a meaningful speed advantage, such as a 16%-speed advantage for a configuration having 53-microseconds per line unblanked period and 63-microseconds per line blanked plus unblanked periods. This buffer memory feature is relatively independent of the type of memory chips available and relatively independent of the scanout and re-addressing

arrangement.

Memory architectural features of sequentially accessing a scanline of pixels and buffering the scanline of pixels for multiple operations at different times for each pixel, such as kernel processing, provide a memory bandwidth enhancement. These features utilize several characteristics of the experimental configuration. Pixels can be accessed from image memory in an ordered form, such as in a scanline form consistent with the ordered form of pixels in a raster scan CRT monitor. The number of pixels to be buffered may be a relatively small percentage of the total image memory capacity, such as 3-scanlines out of 512-scanlines, about 1/2% of the image memory capacity.

The memory configuration in the experimental display system has significant advantages over other memory architectures. For example, it provides more than a 10-MHz rate for a kernel of 9-pixels, which is equivalent to a pixel access rate of more than 90-MHz (10-MHz kernels times 9-pixels/kernel). It performs this function using 200-ns RAM chips, implying only a 5-MHz pixel rate ( $1/200\text{-ns} = 5\text{-MHz}$ ). Therefore, it provides about an 18-times advantage in memory access rate ( $90\text{-MHz}/5\text{-MHz} = 18\text{-times}$ ).  
F Alternately, accessing each of the kernel pixels at the implied 5-MHz access rate provides a kernel access rate of 0.55-MHz (5-MHz/9-pixels) instead of the 10-MHz kernel rate. Such an 0.55-MHz rate may be further degraded because the 200-ns access time of the RAM may be degraded by additional propagation delays, such as address input and data output logical propagation delays. Consequently, an advantage of about 20-times in memory bandwidth can be obtained.

Speed advantages can be obtained by accessing information from memory at a higher duty cycle and lower clock rate and outputting information at a lower duty cycle and higher clock rate by buffering accessed information with a buffer memory. For example, in a display system; pixels can be accessed from image memory at a slower clock rate during the full line period, blanked and unblanked line period portions, and pixels can be output to the display interface at a higher clock rate during the unblanked portion of the line and not during the blanked portion of the line.

Important speed and memory access advantages can be obtained by buffering of information that will be used a plurality of times, such as by buffering pixel information that will be used 9-times for a 9-kernel pixel. In this configuration, each pixel can be used to generate 9-output pixels, where a pixel will be used at each of 9-kernel positions for generating 9-adjacent pixels. Therefore, buffering the output pixels can reduce memory access bandwidth by a factor related to the number of times the pixel is used, 9-times in this example.

A system requiring a 13-MHz pixel rate during the unblanked period of a line and requiring accessing of 9-pixels from image memory for each output pixel implies a memory access rate of 117-MHz (13-MHz times 9-input pixels for each output pixel). However, accessing of memory during the full line period (both blanked and unblanked portions) can provide an improvement of 1.24-times by accessing of pixels at about a 10.5-MHz rate during the full line period, blanked and unblanked portions, (13-MHz times 53-

microseconds unblanked period divided by 67-microsecond for the blanked and unblanked period). Further, the buffer memory arrangement discussed herein overcomes the need to access 9-pixels from image memory to generate a single output pixel with a novel buffer memory configuration. Consequently, the buffer memory arrangement discussed herein can provide an improvement over 11-times (1.24 times 9) over a non-optimized arrangement.

### FIFO Buffer Memory Architecture

The arrangement discussed above for generating a sequential line of pixels can be adapted to provide a triple line of pixels for spatial processing. A single line of pixels can be converted to a triple line of pixels with a look-ahead buffering arrangement. For example, if the memory map scanout is performed a line ahead of the display refresh and if 3-scanlines of scanned-out pixels are buffered, then the 9-pixels for a particular kernel, which is centered about the present pixel being displayed, is available in the 3-scanline buffer. The 3-scanline buffer can be implemented in various arrangements, such as with a double buffer arrangement and with a FIFO arrangement. A FIFO arrangement will now be discussed with reference to Fig 5D.

The arrangement shown in Fig 5D comprises 3-scanline buffers 513H implemented as FIFOs and an output register array 513A implemented to provide a 9-pixel kernel in parallel for spatial processing. The FIFOs are shown connected in serial form so that the next line of pixels being accessed are shifted into the lower FIFO channel 513E the present line of pixels, which is being shifted out of the lower FIFO channel 513E, is recirculated and shifted into the middle FIFO channel 513D; and the present line of pixels, which is being shifted out of the middle FIFO channel 513D, is recirculated and shifted into the upper FIFO channel 2610C. Consequently, the lower FIFO channel 513E contains the next line of pixels, the middle FIFO channel 513D contains the present line of pixels, and the upper FIFO channel 513C contains the prior line of pixels. The parallel shifting of all three FIFO channels 513H makes available the next line, present line,

and prior line of pixels; which can be shifted into the parallel register array 513A for spatial processing with processor 513B.

FIFOs can be implemented with RAMs, where the next address to be loaded is generated with a write address counter <sup>513I</sup> ~~513I and~~ and the next address to be accessed is generated with a read address counter 513F. As pixels are shifted into the FIFO, the write address counter 513I is incremented to the next available FIFO address. As pixels are shifted out of the FIFO, the read address counter 513F is incremented to the next FIFO address to access the next pixel in sequence. The read address counter can be incremented at a rate consistent with the display refresh requirements, which can be a constant rate along a scanline and a zero rate during line retrace and frame retrace. The write address counter can be incremented at a rate consistent with the scanout of pixels from the image memory, which can be a constant rate within a block and a lower rate between blocks. Therefore, the FIFO permits interfacing of two circuits operating asynchronously relative to each other. Comparitor <sup>A</sup> 513G can be implemented to compare the read address and the write address from read address counter 513F and from write address counter 513I, respectively. In normal operation, the write address is ahead of the read address. However, if the FIFO gets overloaded, the read address counter can be advanced up to and past the write address counter; thereby overwriting important data. Comparitor <sup>A</sup> 513G detects such an overwrite condition and generates an error signal 513J, such to alert the operator or to override input operations.

The line buffer can be initialized by initiating image memory scanout prior to the first line to be displayed in order to load the 3-lines into the line buffer. This can be achieved with a look-ahead executed during the end of the previous frame, such as during the frame retrace period. Also, the system can be configured so that the average scanout rate from image memory is greater than the refresh rate of the display. Therefore, the line buffer will have sufficient pixels for refreshing the display. If the line buffer becomes fully loaded, detected by the read address approaching the write address; a disable signal can be generated to disable scanout from image memory until the display refresh has vacated capacity in the line buffer.

The parallel registers 513A can be implemented in a 3-pixel by 3-pixel array so that each row of three registers receives 3-pixel words from the corresponding line buffer channel. For example, the top row of registers 0, 1, and 2 receives prior pixel words from the top line buffer channel; the middle row of registers 7, 8, and 3 receives present pixel words from the middle line buffer channel; and the bottom row of registers 6, 5, and 4 receives next pixel words from the bottom line buffer channel. Implementation of the parallel registers to transfer in the row direction toward the right permits pixel words to be shifted into the register rows from the line buffer and to be shifted along the register row toward the right as the scanline progresses. Therefore, only a single pixel need be shifted out of each line buffer channel for a particular pixel period; where the three rows of pixels are shifted toward the right to cause the kernel to shift toward the right along the scanlines.

The intensities for the 9-pixels stored in registers 513A can be output to the weighting multipliers and adders 513B for sum of the product processing for spatial filtering.

### Kernel Memory Operations

Image processing may involve processing of a plurality of pixel intensities to generate a single pixel intensity for display. For example, an output pixel to a display monitor may be a convolved sum of the weighted intensities of a kernel of pixels. A kernel of pixels may be a 3-pixel by 3-pixel kernel, a 5-pixel by 5-pixel kernel, or other kernels. The need to process multiple pixels to generate each output pixel implies that image memory must be accessed a plurality of times for a plurality of pixels that are to be convolved for each output pixel. For example, in a configuration that processes a 9-pixel kernel to generate an output pixel, image memory would be accessed 9-times for a 9-pixel kernel for each output pixel period. This imposes a memory bandwidth requirement of 9-times the output pixel rate for the image memory access rate. Therefore, a system using such a configuration would require very high speed image memories or alternately would have relatively low resolution; either spatial resolution, or temporal resolution, or both; or alternately would have a combination of high image memory speed requirements and low output resolution. For example, a system having a 10-MHz output pixel requirement and a 9-pixel kernel spatial filtering output processing requirement would appear to need a 90-MHz memory pixel access rate. However, 90-MHz RAM chips may have poor availability and may be relatively expensive compared to slower speed RAM chips.

An arrangement is provided herein using a buffer memory to store pixels accessed from image memory at the output pixel rate and providing a kernel of 9-pixels to the output filter. This

configuration permits accessing of a 9-pixel kernel at more than a 10-MHz rate, yielding an equivalent pixel rate of more than 90-MHz (9-pixel kernel times 10-MHz). However, this configuration uses RAM chips having a 200-ns access time, implying an equivalent pixel access rate of about 5-MHz. This yields an improvement of about 18-times (90-MHz/5-MHz) in memory access bandwidth and permits use of less expensive lower speed RAMs to obtain relatively high resolution and to support multiple pixel kernel spatial filtering.

### Blanked and Unblanked Memory Operations

A video signal can be generated with a plurality of line synchronization signals having a blanked portion for horizontal retrace and having an unblanked portion for displaying a line of the image. For example, the line rate may be about 15,750-Hz; yielding about a 63-microsecond line period. Approximately 10-microseconds of this line period may be blanked, yielding about a 53-microsecond unblanked line period. In a system having a direct output from the image memory to the display monitor, the image memory may be accessed during the unblanked period and the image memory may not be accessed during the blanked period. However, important advantages can be obtained by accessing image memory during the full line period, during both blanked and unblanked portions of the line period. This can be implemented with a buffer memory that stores pixels during the blanked and unblanked portions of the line period and accesses pixels for display during the unblanked portion of the line period.

The number of pixels that are input to the line buffer from image memory may be the same as the number of pixels that are output from the line-buffer to the display monitor. The output period can be implemented to be the unblanked portion of the line period and the input period can be implemented to be the blanked plus unblanked portions of the line period. Therefore, because the output period is shorter than the input period; the output rate can be greater than the input rate. Consequently, the image memory generating the input rate can be slower than the output pixel rate and, hence, the output pixel rate can be greater than indicated by image memory speed. For example, in a system having

a 63-microsecond line period and a 53-microsecond unblanked portion of the line period, a 10-MHz buffer input rate can be buffered to provide about a 12-MHz output rate because of the greater input period for loading the buffer; i.e., 63-microseconds; compared to the output period for unloading the buffer; i.e., 53-microseconds. This permits relatively slower speed RAMs to be used for image memory, which may be a larger memory (i.e.; a 265,000 pixel image memory) in conjunction with a higher speed buffer memory, which may be a smaller memory (i.e.; a 512-pixel line buffer memory); thereby efficiently providing high pixel rates with relatively slow speed image memory RAMs.

## 4/3rds Buffer Memory Architecture

The buffer memory features discussed herein are illustrated with a novel buffer memory, herein called a 4/3rds buffer memory, that provides double buffer memory capabilities. A double buffer memory duplicates the buffer memory requirements so that a first buffer memory can be in the process of being loaded while a second buffer memory is in the process of being unloaded. The 4/3rds buffer memory is more efficient than a full buffer memory because it replicates only 1/3rd of the redundant buffer memory; where 3-lines are being simultaneously unloaded while a 4th-line is being loaded. This 3-line multiple buffer, implemented as a 4/3rds line buffer, is illustrative of other multiple buffer type configurations that are optimized to reduce the amount of memory from double to less than double. For example, the improved multiple buffer memory can be implemented in the form of a 3/2's buffer, a 5/3rds buffer, and other optimized multiple buffer configurations.

The optimized multiple buffer is exemplified with a precession technique. For example, a fractional portion of the buffer is replicated for loading while a full buffer is being unloaded. Then, the newly loaded portion of the buffer is precessed into the output buffer, the output buffer is precessed within the output buffer, and part of the output buffer is precessed to the input buffer. For example, in the 4/3rds buffer arrangement discussed herein; line buffers, line buffer-1 to line buffer-4, are shared between input and output buffers; where the input line buffer uses a 1-line buffer and the other line buffer uses a 3-line buffer. Line-buffers, line buffer-1 to line

buffer-4, may be precessed in sequence; where each line-buffer is sequentially switched to the input line buffer, output line-buffer channel-1, output line-buffer channel-2, output line-buffer channel-3, and then back to input line-buffer in sequence. When line buffer-1 is in the input line buffer position, line buffer-2 is in the channel-1 position, line buffer-3 is in the channel-2 position, and line buffer-4 is in the channel-3 position. During the next line operation, line buffer-1 is switched to the channel-1 position, line buffer-2 is switched to the channel-2 position, line buffer-3 is switched channel-3 position, and line buffer-4 is switched to the input buffer position. The 4-line buffers precess in sequence through the input buffer; output buffer channel-1, output buffer channel-2, and output buffer channel-3, and input buffer positions. Consequently, a new line of information is being loaded into the line-buffer in the input buffer position, the most recent line of information is being output in the output buffer channel-1 position, the next most recent line of information is being output in the output buffer channel-2 position, and the oldest line of information is being output in the output buffer channel-3 position. Hence, output buffer channel-1 provides kernel pixels 1, 2, and 3; output buffer channel-2 provides kernel pixels 8, 9, and 4; and output buffer channel-3 provides kernel pixels 7, 6, and 9. For the next line condition, each line is shifted end around; where the most recent line of information loaded in the line-buffer in the input position is switched to the output buffer channel-1 position, the line-buffer in the

output buffer channel-1 position is switched to the output buffer channel-2 position, the line-buffer in the output buffer channel-2 position is switched to the output buffer channel-3 position, and the line-buffer in the output buffer channel-3 position (which is the oldest information and is no longer needed) is switched to the input buffer position for loading of a new line of information. This end-around shifting of lines of information causes a precession of the 4-line buffers to the 4-buffer positions to provide for inputting of a new line of information and for simultaneously outputting each of the last 3-lines of information for implementing a parallel 3-line 9-pixel kernel for spatial filtering.

### Other Buffer Configurations

Various configurations of the buffer of the present invention have been discussed above to illustrate how the various features and devices of the buffer of the present invention can be used to implement a system. These configurations are illustrative of a large number of other configurations that can be implemented from the teachings herein.

The buffer of the present invention has been described in combination with other novel features of the present invention, such as a novel memory architecture. However, the buffer of the present invention can be used with other systems having other configurations, where the buffer of the present invention is not constrained to use only with the other elements of the system of the present invention.

The buffer of the present invention has been described for implementation with integrated circuit RAMs. Alternately, the buffer teachings of the present invention can be implemented with other memories; such as CCD memories, core memories, and other types of memories. Also, many of the teachings associated with the buffer memory of the present invention can be utilized with ROMs.

F  
F  
<sup>15</sup> The weight memory of the present invention ~~has been~~  
<sup>herein</sup>  
described for implementation with integrated circuit RAMs.

Alternately, the weight memory teachings of the present invention can be implemented with other memories; such as CCD memories, core memories, ROMs, PROMs, EROMs, and other types of memories. For example, the weight memory can be implemented as a weight ROM in place of the weight RAM.

The buffer of the present invention has been discussed in the form of a 4/3-buffer as an improvement on a double buffer. Other non-integer or fractional multiple buffers can be implemented, such as a 5/3-buffer instead of a 4/3-buffer, for replacing other integer multiple buffers, such as a triple buffer instead of a double buffer.

The buffer of the present invention has been discussed in conjunction with a 9-pixel kernel spatial filter. However, the buffer of the present invention can be implemented with other post processor arrangements with a kernel processor and can be implemented with other kernel processors, such as a 25-pixel kernel or a 48-pixel kernel processor.

EXPERIMENTAL SYSTEM

## EXPERIMENTAL SYSTEM ARCHITECTURE

### General Description

An experimental system is configured with a host computer, a display processor, <sup>A</sup> and memory, and a color monitor. The host computer is implemented with an S100 bus configuration using S100 compatible boards, such as Compupro boards; together with disk drives, printers, and other peripherals. The CRT monitor is a conventional color monitor having an analog RGB input, shown with monitor documentation included herewith. The display processor and memory arrangement is configured with a plurality of wire wrap boards. These boards include a logic board, BL1; 2-memory boards, BM1 and BM2; a rear-end board, BR1; and a buffer board, BB1.

Operation of hardware and software in the experimental system discussed herein in conjunction with a color monitor demonstrates operation of the system, meeting of system objectives, and providing actual reduction to practice. For example, information has been loaded into image memory and has been display processed and displayed to demonstrate operation.

### Supervisory Processor Interface

The interface between the supervisory processor and the display processor comprises input synchronization signals from the display processor to the supervisory processor to synchronize the supervisory processor with the display processor operations and output commands from the supervisory computer to initialize the display processor.

Synchronization signals include a frame synchronization signal and a line synchronization signal. The frame synchronization signal occurs during vertical retrace and vertical blanking of the video signal. The line synchronization signal occurs during horizontal retrace and horizontal blanking of the video signal. An interlaced scan arrangement is used for the experimental system, although other scan arrangements can readily be accommodated. A field identification signal is provided that identifies whether the field is a first field or a second field of the interlaced scan.

Communication between the supervisory processor and the display processor is performed with a 3-port parallel interface to a Compupro Interfacer-II board under program control. Each port has 8-parallel input lines and 8-parallel output lines. The port assignments are listed in the COMPUTER PORT TABLE included herein. Output signals are defined as DO signals and input signals are defined as DI signals. Port identification; A, B, or C; follows the DO or DI symbol. Signal line identification follows the port identification; i.e., DOA7 identifies output line-7 in port-A.

Signal DOA5 controls loading in sequential and random access form. Sequential loading is selected when DOA5 is high and random access loading is selected when DOA5 is low. When DOA5 is high, an output strobe on DOA7 causes the pixel address registers to be updated with the related slope parameters on the falling edge of the strobe. This insures compatibility with the DOA7 strobe used as a write-bar signal to write the previous pixel information into the previously addressed pixel in image memory.

Signal DOA6 controls loading and running operations. Running is selected when DOA6 is high and loading is selected when DOA6 is low. In general, DOA6 is high during displaying of images and DOA6 is low during loading of images into memory.

Signal DOA7 strobes the information output with Port-B and Port-C into the destinations. DOA7 is normally low, <sup>and</sup> DOA7 is pulsed high and then pulsed low under program control to form an output strobe.

Signal DIA0 inputs the frame sync pulse from the CRT monitor interface. This frame sync pulse is the blanking pulse that blanks the CRT monitor during the vertical retrace period and during a predetermined number of lines prior to and subsequent to the vertical retrace period. This frame sync pulse occurs once per field, twice per frame, in the interlaced scan system as implemented with the demonstration system. The rising edge of the frame sync pulse, detected under program control, initiates loading of the parameters for a new field from the supervisory processor into the display processor.

Signal DIA2 inputs the line sync signal from the CRT monitor interface, which is implemented for hardware control but not software control in the experimental system.

Signal DIA4 inputs the frame identification signal from the CRT monitor interface. DIA4 is high during the field-1 period and low during the field-2 period.

Signals DOB0 through DOB7 output the information to be loaded into the destination identified with Port-C. This information can be delta information to be loaded into the delta registers, pixel address information to be loaded into the pixel address registers, and pixel data to be loaded into image memory.

Signals DOC0 through DOC7 output the address of the destination to be loaded with the Port-B output signals. The various destinations are listed in the TABLE OF DESTINATION SELECT ASSIGNMENTS included herein; including the 4-delta registers each having an MSH and an LSH, 2-address registers each having an MSH and an LSH, data to be written into image memory, and weights to be written into a weight table memory.

### Image Loading

Loading of an image into memory is performed by loading the XP and YP-address registers with the address of each pixel to be loaded, then outputting the pixel information to be loaded with Port-B, and then strobing the pixel information into image memory with the DOA7 signal. A sequential load feature is provided under control of the DOA5 signal. When the DOA5 signal is high, a vector can be loaded; where the previously loaded pixel address is incremented with the related delta parameter to obtain the next pixel address to reduce software overhead and thereby speedup loading of image memory.

Loading of image memory with the supervisory processor is performed with a 3-port output arrangement having 8-bits per port. The first port, Port-A, communicates control signals between the supervisory processor and the display processor. The second port, Port-B, communicates address and data information to be loaded into the display processor between the supervisory processor and the display processor. The third port, Port-C, selects the register or memory in the display processor for loading. The protocol involves outputting of the destination address on Port-C, outputting of information to be loaded into the display processor on Port-B, and then outputting of a data strobe on Port-A. The data strobe loads the output information into the selected destination.

A program to load vectors into memory is provided herein as the BASIC PROGRAM LISTING LD.ASC and is briefly discussed in the section entitled Software herein.

### Software

Programs have been developed to operate the experimental system and are included herein in the tables BASIC PROGRAM LISTINGS. These programs are source programs, that are compiled with a Basic compiler and linked with a Basic linker to obtain compiled Basic programs. Compiled Basic programs run significantly faster than interpretive Basic programs, which maintains real time synchronization between the display processor and the supervisory processor. The source listings may be readily compiled and linked by one skilled in the art to provide the compiled Basic programs executed to perform the image loading and image processing operations. The programs are programmed to be menu driven, prompting the operator to select various operator-selectable options.

The Basic listings included herein have extensive annotation to teach one skilled in the art the features implemented therein.

The BASIC PROGRAM LISTING LD.ASC provided herein teaches loading of vectors into memory. The BASIC PROGRAM LISTING GRAPH.ASC provided herein teaches refreshing of a CRT monitor from memory. These listings are clearly coded and amply annotated to teach one skilled in the art how to operate the experimental system disclosed herein under program control.

### Description of DIS.ASC Listing

One configuration of the system of the present invention is implemented with a computer executing a program, shown in the BASIC PROGRAM LISTING DIS.ASC provided herein, once per field to load initial conditions into the geometric processor. This program generates initial conditions for the iterative processing and then iteratively generates conditions for the geometric processor. The iterative processing is performed once per field. It is implemented with multiple field interpolation which generates the initial conditions for an 8-field period and then interpolates <sup>1</sup>  
~~in~~<sup>^</sup> between the 8-field period to obtain the initial conditions for each field. This may be considered to be a <sup>t</sup>  
~~temporal~~<sup>^</sup> domain interpolation, in contrast to <sup>s</sup>~~spacial~~<sup>^</sup> domain interpolation; where temporal domain interpolation interpolates the initial conditions between temporally sequential fields rather than spatially within a field.

The BASIC PROGRAM LISTING DIS.ASC provides the detailed Basic program code for one configuration of the display program. A detailed description of the code in the BASIC PROGRAM LISTING DIS.ASC is set forth below. This detailed description is provided with the pertinent lines of code replicated and indented and preceded by a text description of these preceding lines of code. The line numbers for these lines of code inbedded in the text provide a cross reference to the correspondingly numbered lines of code in the BASIC PROGRAM LISTING DIS.ASC provided herein. It should be noted that the Basic compiler and interpreter presently in use interprets an apostrophe "'" as terminating the line of code. Consequently any text to the right of an apostrophe is not

executed by the computer. For example, the apostrophe permits  
annotations to be entered without concern for interfering with  
executable code.

Terminology used in the BASIC PROGRAM LISTING DIS.ASC is  
briefly defined in the DIS.ASC TERMINOLOGY TABLE and is discussed  
in detail thereafter.

## DIS.ASC TERMINOLOGY TABLE

K2\$ => System configuration

M => Murphy

C => Camille

DR => Degree to radian conversion factor

RD => Radian to degree conversion factor

A% => Joystick loop counter parameter

B% => Joystick channel parameter

C% => Joystick input parameter

D% => Joystick channel parameter

E% => Joystick calibration status

PR7\$ => Select printouts

PR9\$ => Select update, each field or alternate fields

PR10\$ => Select, clear remainders or don't clear remainders

PR11\$ => Select slope calculation, using CINT roundoff or not  
using CINT roundoff

PR12 => Scale factor minimum value

PR13 => Scale factor maximum value

PR16\$ => Select, integer or non-integer processing

PR21\$ => Select initial point and slopes, fractional or  
non-fractional

DB1% => Select deadband

TS1 => Reciprocal of number of line slices

DIS.ASC TERMINOLOGY TABLE (CONTINUED)

JSXV% => Joystick X-translational input in viewport coordinates  
          offset to null at 128

JSXB% => Joystick X-translational input outside of deadband

JSXK% => Joystick partially resolved X-translation parameter

JSYK% => Joystick partially resolved Y-translation parameter

JSYV% => Joystick Y-translational input in viewport coordinates,  
          offset to null at 128

JSYB% => Joystick Y-translational input outside of deadband

JSS% => Joystick scaling input, offset to null at 128

JSA% => Joystick rotational input, offset to null at 128

JSX% => Joystick X-translational input, offset to null at 128

JSY% => Joystick Y-translational input, offset to null at 128

BX5% => Joystick X-translation calibration parameter

BY5% => Joystick Y-translation calibration parameter

BA5% => Joystick angle calibration parameter

BS5% => Joystick scale factor calibration parameter

JSSB% => Biased JSS%

JSAB% => Biased JSA%

XC1, YC1 => Viewport center coordinate in eighth pixels

KSAR => Sine of angle AR

KCAR => Cosine of angle AR

DIS.ASC TERMINOLOGY TABLE (CONTINUED)

TX => Center of rotation offset from center of viewport in  
X-direction

TY => Center of rotation offset from center of viewport in  
Y-direction

X5V => Viewport X-dimension to pixel resolution

Y5V => Viewport Y-dimension to pixel resolution

X5I => Image memory X-dimension to pixel resolution

Y5I => Image memory Y-dimension to pixel resolution

XSV => Viewport, aspect ratio (1.58)

XS => X-axis scale factor (with viewport aspect ratio) to  
1/256 pixel resolution

YS => Initial Y-axis scale factor to 1/256 pixel resolution

YSS => Y-scale factor parameter

SXXV => X-scale factor parameter

XSVR => Reciprocal of XS

R1 => Diameter of viewport to pixel resolution

KS1% => Image memory X-dimension to eighth pixel resolution

KS2% => Image memory Y-dimension to eighth pixel resolution

Q2 => Center of rotation coordinate as offset in Y from the  
center of the viewport to pixel resolution

Q3 => Center of rotation coordinate as offset in X from the  
center of the viewport in pixels

AP1 => Angle of R1 from the horizontal in radians

DIS.ASC TERMINOLOGY TABLE (CONTINUED)

KB1, KB2 => Intermediate viewport variable to eighth pixel  
resolution

KF4, KF5, KF6, KF7 => Intermediate variables to eighth pixel  
resolution

XIP1 => X-initial point coordinate for the first field

*F* *B*  
XIP1N => ~~Buffered~~ X-initial point coordinate for the first field

YIP1 => Y-initial point coordinate for the first field

YIP1 => Buffered Y-initial point coordinate for the first field

XIP2 => X-initial point coordinate for the second field

XIP2N => Buffered X-initial point coordinate for the second field

YIP2 => Y-initial point coordinate for the second field

YIP2N => Buffered Y-initial point coordinate for the second field

XR% => X-row slope

XRN% => Buffered X-row slope

YR% => Y-row slope

YRN% => Buffered Y-row slope

XP% => X-pixel slope

XPN% => Buffered X-pixel slope

YP% => Y-row slope

YP%N => Buffered Y-row slope

DIS.ASC TERMINOLOGY TABLE (CONTINUED)

DXIP1% => Delta XIP1%

XYIP1% => Delta YIP1%

DXIP2% => Delta XIP2%

DYIP2% => Delta YIP2%

DXP% => Delta buffered XP

DYP% => Delta buffered YP

DXR% => Delta buffered XR

DYR% => Delta buffered YR

DXIP1R% => Delta XIP1 remainder

DYIP1R% => Delta YIP1 remainder

DXIP2R% => Delta XIP2 remainder

DYIP2R% => Delta YIP2 remainder

DXPR% => Delta XP remainder

DYPR% => Delta YP remainder

DXRR% => Delta XR remainder

DYRR% => Delta YR remainder

CA1% => Y-row position MSH for field-1

CB1% => Y-row position LSH for field-1

CC1% => X-row position MSH for field-1

CD1% => X-row position LSH for field-1

DIS.ASC TERMINOLOGY TABLE (CONTINUED)

CA2% => Y-row position MSH for field-2

CB2%=>Y-row position LSH for field-2

CC2% => X-row position MSH for field-2

CD2% => X-row position LSH for field-2

XPM% => X-pixel slope MSH

YPM% => Y-pixel slope MSH

XRM% => X-row slope MSH

YRM% => Y-row slope MSH

XPL% => X-pixel slope LSH

YPL% => Y-pixel slope LSH

XRL% => X-row slope LSH

YRL% => Y-row slope LSH

The following code prints the file name and revision date on the terminal for verification by the operator of the configuration being executed.

```
200 PRINT: PRINT: PRINT "FILE: DIS.ASC, REV. 10/7/4 19:00"
```

The following code clears the program and zero sets the program parameters.

```
240 CLEAR
```

The following code selects the system that is being used, where the Murphy system and the Camille system are both used with the image processor and where the Murphy system and the Camille system have different keyboard addresses.

```
260 INPUT "MURPHY (M) OR CAMILLE (C)";K2$ 'SELECT  
SYSTEM CONFIGURATION
```

The following code is used to calibrate the joysticks. The FOR NEXT loop sequentially accesses each of the four analog to digital converters (ADCs) associated with four joystick potentiometers and displays the four ADC parameters on the terminal. This code is executed iteratively to provide the operator with a continually updated readout on the terminal joystick bias controls. When the operator is satisfied with the calibration, he depresses a keyboard switch to exit the joystick calibration routine.

```
280 PRINT: PRINT "JOYSTICK BIAS VALUES" 'CALIBRATE JOYSTICK:  
300 E% = 0
```

The following code iteratively steps through the four joystick addresses to sequentially access each of the four joystick parameters for display to the operator. The STEP 2 operation makes the code consistent with the port addresses that

are packed into bit positions of the output word having a binary weighting of a factor of two. The  $B\% = A\%-2$  operation establishes the port addresses referenced to zero instead of being referenced to unity. The output of the  $B\%$  parameter with the OUT instruction selects one of four channels in the ADC multiplexer. The  $C\%$  parameter is the input parameter from the selected ADC. The GOSUB operation calls the subroutine that processes the input ADC parameter, which is described in detail below.

```
320 FOR A\% = 2 TO 8 STEP 2: B\% = A\%-2: OUT 236,B\%: C\% = INP  
(237): GOSUB 440  
340 NEXT A\%
```

The following code tests for an operator command to exit the joystick calibration operation. A test is made of the K2\$ parameter to determine if the Murphy system or Camille system is being used. For the Murphy system, input port-1 is tested. For the Camille system, input port-93 is tested. The input parameter is masked to isolate the status bit and the status bit is tested as indicative of a keyboard operation. If the status bit is 0-set, the keyboard operation has not been performed and the iterative processing for joystick calibration is continued. If the status bit is 1-set, the keyboard operation has been performed and the iterative processing for joystick calibration is exited.

```
360 IF K2$ = "C" THEN 400  
380 A8\% = INP (1): GOTO 420 'KEYBOARD EXIT OF JOYSTICK CALIBR.  
400 A8\% = INP (93)  
420 A8\% = A8\% AND 2: IF A8\% = 0 THEN 320 ELSE 660
```

The following code is a subroutine that prints the joystick function and the related joystick parameter on the terminal to display a single line of four joystick parameters. The D% parameter is calculated, selecting one of four joysticks. The ON GOTO instruction selects the one of four PRINT instructions based upon selected joystick, D%. The commas after the PRINT instructions for three parameters preempt a line return. The absence of a comma after the Y-parameter PRINT instruction causes a line return, which starts each new line with a Y-parameter.

```
440      ****
460      'SUBROUTINE TO ACQUIRE JOYSTICK VALUES
480      D%=A%/2: ON D% GOTO 500,520,540,580
500      PRINT "SCALE=";C%,: BS5%=C%: GOTO 620
520      PRINT "X="      ;C%,: BX5%=C%: GOTO 620
540      PRINT "ANGLE=";C%,: BA5%=C%: GOTO 620
580      PRINT "Y="      ;C% : BY5%=C%
600      PRINT CHR$(11);
620      RETURN
640      ****
```

The following code sets up initial conditions. These initial conditions can be fixed, as established by the equality statements, or alternately can be input by the operator from a menu with the INPUT statements. PR7\$ defines a printout command that is no longer used. PR9\$ defines whether interpolation is performed for each field or for each frame (alternate fields), providing either 8-interpolation operations for 8-field periods or 4-interpolation operations for 8-field periods respectively.

F PR7\$ is used to evaluate effects of interpolation period or ^

frequency. PR10\$ defines whether the remainders for the interpolation accumulators are cleared for each interpolation iteration to experiment with clearing of remainders and preserving remainders between interpolative iterations. PR11\$ defines whether the slope parameters are rounded off to integers with the CINT instruction or are processed without the CINT instruction to experiment with different ways of processing the slope parameters. Operator selection of PR7\$ to PR11\$ has been disabled and fixed conditions have been defined with equality statements.

```
680  PR7$="N"          ' INPUT "PRINTOUT? 'Y' OR 'N'" ;PR7$  
700  PR9$="Y"          ' INPUT "UPDATE EACH FIELD, NOT  
                         ALTERNATE FIELDS; 'Y' OR 'N'" ;PR9$  
720  PR10$="Y"         ' INPUT "CLEAR REMAINDERS; 'Y' OR  
                         'N'" ; PR10$  
740  PR11$="Y"         ' INPUT "SLOPE USING CINT  
                         ROUNDOFF; 'Y' OR 'N'" ;PR11$
```

The following code permits the operator to select the minimum and maximum scale factors for expansion and compression.

```
760  INPUT "SCALE FACTOR; MIN, MAX; .1 TO 1500; 10,600  
                         IS NOMINAL";PR12,PR13
```

The following code permits the operator to select the nominal scale factor maximum and minimum values of PR12 =10 and PR13=600 by entering a 0,0 for PR12, PR13.

```
780      IF PR12=0 OR PR13=0 THEN 800 ELSE 820  
800      PR12=10: PR13=600
```

The following code tests for the interpolative iterations selected and calculates the time scale factor, TS1, as a function of the number of interpolative updates per each 8-field interpolative iteration. Use of this time scale factor is discussed with reference to lines 3260 to 3320 hereinafter.

```
820 IF PR9$="Y" GOTO 860 'INTERPOLATE ONCE EACH  
FIELD OR ONCE EACH FRAME
```

```
840 TS1=1/4: GOTO 880 'NUMBER OF TIME SLICES  
860 TS1=1/8
```

The following code permits the operator to select one of two window geometries, the new or the old window geometry. The old window geometry is discussed with reference to Figs 2C and 2D. The new window geometry is discussed with reference to Figs 2E and 2F.

```
880 INPUT "NEW WINDOW GEOMETRY: 'Y' OR 'N'" ; PR32$
```

The following code permits the operator to select either integer or non-integer translation. Integer translation reduces scintillation but pre-empts subpixel operations. Non-integer translation increases scintillation and permits subpixel operations.

```
900 PR16$="F" 'INPUT "INTEGER (I) OR NON-INTEGER  
(F) TRANSLATION" ; PR16$
```

The following code permits the operator to select the offset of the center of rotation as a function of the window geometry previously selected. For the old window geometry, TX and TY are input. For the new window geometry, X1 and Y1 are input. These parameters provide the X-offset and the Y-offset respectively of the center of rotation as offset from the X1 and Y1 coordinates.

```
920 IF PR32$="Y" GOTO 960
940 PRINT: INPUT "OFFSET CENTER OF ROTATION:
TX,TY";TX,TY: GOTO 980
960 PRINT: INPUT "OFFSET CENTER OF IMAGE: X1, Y1" ;X1,Y1
```

The following code sets up initial conditions. These initial conditions can be fixed, as established by the equality statements, or alternately can be input by the operator from a menu with the INPUT statements. DB1% defines the magnitude of the deadband for the joysticks. The program does not respond to a joystick input within the deadband of the joystick to compensate for joystick bias and offset. PR21\$ defines whether fractional or integer values for the initial point and slope parameters are used. Fractional values are used for subpixel processing, such as to provide smooth motion. Integer values are used for testing,  
such for scintillation testing.

```
980 DB1%=20      'PRINT: INPUT "DEADBAND SELECTION;
20 IS NOMINAL";DB1%
1000 PR21$="Y"    'PRINT: INPUT "FRACTIONAL INITIAL
POINT AND SLOPES, Y OR N";PR21$
```

The following code establishes conversion factors for degrees to radians (DR) and for radians to degrees (RD).

```
1020 DR=.017453292#: RD=57.29577951000028#
```

The following code defines the viewport dimensions; which are 512-pixels per line (X5V) and 482-lines per frame (Y5V) for this implementation.

```
1040 X5V=512: Y5V=482      'VIEWPORT DIMENSIONS
```

The following code is an annotation that describes how the X5V parameter is derived as a function of the image processor pixel rate.

```
1060  '** ADAPT X5V FOR PIXEL RATE; (54 US) (9.15  
MHZ) (16/18 CLOCK WIDTH)
```

The following code defines the image memory dimensions; which are 512-pixels per line (X5I) and 512-lines per image (Y5I) for this implementation.

```
1080  X5I=512: Y5I=512 'IMAGE MEMORY DIMENSIONS IN PIXELS
```

The following code represents calculations for geometric window relationships. These relationships pertain to prior window configurations, which are discussed with reference to Figs 2C, 2D, and 2G. Because these relationships represent prior relationships, they are deleted with apostrophes and maintained as annotations.

```
1100  'A      R1=SQR(X5V^2+Y5V^2)      'UNITS OF PIXELS  
1120  'A      AP1=ATN(Y5V/X5V)  
1140  'A      A6=ATN(X5V/Y5V): AP1=90*DR-A6
```

The following code defines the image memory dimensions in eighth pixel units by multiplying the image memory dimensions X5I and Y5I by 8-eighth pixel units.

```
1160  KS1%=X5I*8: KS2%=Y5I*8 'IMAGE MEMORY DIMENSIONS  
IN EIGHTH PIXELS
```

The following code defines parameters Q2 and Q3 in terms of the viewport dimensions and in terms of the offset of the center of rotation.

```
1180  Q2=(Y5V/2)-TY: Q3=(X5V/2)+TX
```

The following code calculates the window parameters for the selected window geometry.

```
1200 IF PR32$="Y" GOTO 1260  
1220 AP1=ATN(Q2/Q3): R1=2*SQR(Q2^2+Q3^2)  
1240 KB1=R1*8*SIN(AP1)/2: KB2=R1*8*COS(AP1)/2: GOTO 1280  
1260 KB1=Q2*8: KB2=Q3*8
```

The following code defines initial conditions for scaling prior to receipt of joystick commands.

```
1280 DS11=1: JSS% =128
```

The following code calculates the initial conditions for the center of the image memory, XC1 and YC1, relative to the center of the viewport and rescales the center coordinates, XC1 and YC1, and offset parameters, TX and TY, to subpixel units with the multiplication by eight.

```
1300 XC1=(X5I/2+X1)*8: YC1=(Y5I/2+Y1)*8: TX=TX*8: TY=TY*8
```

The following code sets F6% and F7% to fixed values with equality statements. F6% is the time-slice counter initial condition. F7% is no longer used.

```
1320 F6% =1: F7% =0
```

The following code defines the viewport aspect ratio for scaling the image as a function of rotational position. The aspect ratio of 1.58 is a measured value for the particular CRT monitor in use. This aspect ratio parameter is empirically derived for the particular monitor in use. This parameter is used in the X-scaling and Y-scaling processing to compensate for the different scale factors in the X-direction and in the Y-direction due to the unequal aspect ratio. If such compensation was not provided; during rotation, the image would <sup>wobble</sup> ~~wobulate~~ as the

scale factor continuously changed from a minimum vertical and maximum horizontal image memory scale factor at 0-degrees and 180-degrees to a maximum vertical and minimum horizontal image memory scale factor at 90-degrees and 270-degrees. Processing of image scale factors with the XSV parameter is provided hereinafter.

1340 XSV=1.58 'VIEWPORT ASPECT RATIO

The following code defines geometric parameters as a function of the window geometry selected. The parameters XS, YS, YSS, and XSVR are scaling parameters. The parameters KSAR and KSAR represent the cosine of the angle AR and the sine of the angle AR, respectively.

1360 IF PR32\$="Y" GOTO 1420

1380 XS=256\*XSV : YS=256: YSS=YS\*(3.90625E-03):  
XSVR=1/(XSV\*256)

1400 KSAR=0: KCAR=1: XSSV=XS\*XSVR: GOTO 1460

1420 XS=256\*XSV : YS=256: YSS=1

1440 KSAR=0: KCAR=1: XSSV=1

The following code generates 16-iterations through the subroutine starting at line 2220 to initialize parameters before starting iterative processing.

1460 FOR E%=1 TO 16: GOSUB 2220: NEXT E% 'INITIAL  
CONDITION GENERATION

The following code causes the processor to wait for a 0-set vertical sync signal for a field-1 condition to synchronize the processor with the CRT sync generator. This is performed by first testing the vertical sync signal and then testing the field

signal. The vertical sync signal is tested by inputting the appropriate word R%, masking the vertical sync signal bit, and looping back until the vertical sync signal is 0-set with the IF THEN instruction. The field signal is tested by inputting the appropriate word R%, masking the field signal bit, and transferring back to the vertical sync signal test if the field signal is not in the field-1 state with the IF THEN instruction.

```
1480 R%=INP (236): S%=R% AND 1: IF S%=1 THEN 1480  
      'LOCKUP ON VERT.SYNC=1
```

```
1500 R%=INP (236): S%=R% AND 16: IF S%=0 THEN 1480  
      'CHECK FIELD
```

The following code defines the iterative processing entrance point for the first iteration and for each subsequent iteration. For the first iteration, iterative operation commences after the proper vertical sync and field conditions have been established, as discussed above. For each subsequent iteration, operation loops back to this point.

```
1520 'ITERATIVE PROCESSING
```

The following code sets the image processor to the run mode for every iteration through the iterative processing return point.

```
1540 OUT 236,64
```

The following code causes the processor to lock up waiting for a 1-set vertical sync signal as indicative of the end of the prior field and the need to load new initial conditions for the next field. This is performed by testing the vertical sync signal by inputting the appropriate word R%, masking the vertical sync signal bit, and looping back until the vertical sync signal

is 0-set with the IF THEN instruction.

1560 'RESYNCHRONIZATION AND FIELD CONTROL PROCESSOR

1580 R% = INP (236): S% = R% AND 1

1600 IF S% = 0 THEN 1580 'LOCKUP ON VERT.SYNC=0

The following annotation describes that the interlaced scan calculations are to be processed and describes the input binary bits for interlaced scan control.

1620 'INTERLACED SCAN CALCULATIONS

1640 'INPUT BYTE 128 064 032 016 008 004 002 001

1660 ' F2 F1 LS FS

The following code sets the image processor to the load mode to discontinue run operations and to load new field information.

1680 OUT 236,0 'COMMAND LOAD, RUN-BAR

The following code causes the processor to determine the field condition for subsequent outputting information for the appropriate field. The field signal is tested by inputting the appropriate word R%, masking the field signal bit, and transferring to the output routine appropriate for the detected field condition, field-1 or field-2 with the IF THEN ELSE instruction.

1700 R% = INP (236): S% = R% AND 16: IF S% = 0 THEN 1860

ELSE 1720 'CHECK FIELD

The following code outputs the row position parameters; X-axis and y-axis least significant half and most significant half parameters; for the field-2 condition. The pixel position parameters are automatically loaded from the row position registers under hardware control and hence need not be loaded

under program control. The OUT 236,128 and OUT 236,0 instructions generate a pulse that goes from a previously 0-set condition to a 1-set condition, and then back to a 0-set condition to pulse the output parameter defined with the OUT 237 instruction into the register addressed with the OUT 238 instruction.

```
1720          'FIELD-2
1740      'OUTPUT POSITION PARAMETERS
1760      OUT 238, 249: OUT 237, CA2%: OUT 236,128: OUT
236,0          'Y-ROW MSH
1780      OUT 238, 250: OUT 237, CB2%: OUT 236,128: OUT
236,0          'Y-ROW LSH
1800      OUT 238, 246: OUT 237, CC2%: OUT 236,128: OUT
236,0          'X-ROW MSH
1820      OUT 238, 247: OUT 237, CD2%: OUT 236,128: OUT
236,0          'X-ROW LSH
1840  GOTO 1980
```

The following code outputs the row position parameters; x-axis and y-axis least significant half and most significant half parameters; for the field-1 condition. The pixel position parameters are automatically loaded from the row position registers under hardware control and hence need not be loaded under program control. The OUT 236,128 and OUT 236,0 instructions generate a pulse that goes from a previously 0-set condition to a 1-set condition, and then back to a 0-set condition to pulse the output parameter, defined with the OUT 237 instruction, into the register addressed with the OUT 238 instruction.

```
1860          'FIELD-1
1880      'OUTPUT POSITION PARAMETERS
```

```
1900      OUT 238, 249: OUT 237, CA1%: OUT 236,128: OUT  
           236,0      'Y-ROW MSH  
1920      OUT 238, 250: OUT 237, CB1%: OUT 236,128: OUT  
           236,0      'Y-ROW LSH  
1940      OUT 238, 246: OUT 237, CC1%: OUT 236,128: OUT  
           236,0      'X-ROW MSH  
1960      OUT 238, 247: OUT 237, CD1%: OUT 236,128: OUT  
           236,0      'X-ROW LSH
```

The following code outputs the slope parameters; X-axis pixel and row and y-axis pixel and row least significant half and most significant half parameters. This configuration is implemented for the slope parameters to be the same for both fields of the frame. Alternately, the OUT 236,128 and OUT 236,0 instructions generate a pulse that goes from a previously 0-set condition to a 1-set condition, and then back to a 0-set condition to pulse the output parameter, defined with the OUT 237 instruction, into the register addressed with the OUT 238 instruction.

```
1980      'OUTPUT SLOPE PARAMETERS  
2000      OUT 238,242:  OUT 237,XPM%:  OUT 236,128: OUT  
           236,0      'X-PIXEL SLOPE MSH  
2020      OUT 238,245:  OUT 237,YPM%:  OUT 236,128: OUT  
           236,0      'Y-PIXEL SLOPE MSH  
2040      OUT 238,248:  OUT 237,XRM%:  OUT 236,128: OUT  
           236,0      'X-ROW SLOPE MSH  
2060      OUT 238,251:  OUT 237,YRM%:  OUT 236,128: OUT  
           236,0      'Y-ROW SLOPE MSH
```

```
2080      OUT 238,243: OUT 237,XPL%: OUT 236,128: OUT  
           236,0          'X-PIXEL SLOPE LSH  
2100      OUT 238,244: OUT 237,YPL%: OUT 236,128: OUT  
           236,0          'Y-PIXEL SLOPE LSH  
2120      OUT 238,240: OUT 237,XRL%: OUT 236,128: OUT  
           236,0          'X-ROW SLOPE LSH  
2140      OUT 238,241: OUT 237,YRL%: OUT 236,128: OUT  
           236,0          'Y-ROW SLOPE LSH
```

The following code selects the run mode to initiate execution of the next field by the geometric processor and then calls the interfield processing subroutine to perform interpolation processing for the next field and geometric processing for the next interpolation iteration.

```
2160  OUT 236,80  'COMMAND RUN, LOAD-BAR ;PULSE-1  
                  BRACKETING COMPUTATION PERIOD  
2180  GOSUB 2220          'PROCESSING FOLLOWS OUTPUT
```

The following code loops back for the next field iteration, to resynchronize with the frame sync signal, to output the field parameters, and to process another slice of the interpolative and geometric processing.

```
2200  GOTO 1520          'LOOP BACK FOR NEXT FIELD
```

The following code selects the processing to be performed for the current time-slice. The processing operations are divided into eight operations, each of which can be performed in a field period, about 15-milliseconds. The time-slice counter F6% keeps track of the next processing to be performed by being incremented for each field iteration. The ON GOTO instruction selects the next processing operation in sequence under control

of the time-slice counter F6%; transferring to the appropriate processing operation out of the eight processing operations for the present time-slice.

```
2220 *****SUBROUTINE FOR INTERFIELD PROCESSING  
2240 F6% = F6% + 1      'INCREMENT TIME SLICE COUNTER  
2260 ON F6% GOTO 4000, 4640, 4800, 2280, 2820, 3360,  
     3440, 3540
```

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters.

```
2280 *****  
2300 GOSUB 5220: GOSUB 5840  'INTERPOLATE
```

The following code calculates the XIP1 initial point X-coordinate for field-1 as a function of the viewport centerpoint X-position, XC1; the rotational X-offset position, TX; and the geometric window parameters, KF4 and KF6.

```
2320 'CALCULATE INITIAL POINT  
2340 XIP1N% = XC1 + TX - KF6 - KF4  
     'EQUIVALENT TO XIP1N% = XC1 - R1 * 8 * COS(A5%) / 2
```

The following code provides an alternate implementation that detects the need for X-axis wrap-around and controls X-axis wrap-around. The IF THEN ELSE instruction tests if the window XIP1N parameter exceeds the image memory boundaries and therefore determines if wrap-around is needed. If needed, the XIP1N parameter is updated by adding or subtracting the image memory X-dimension, KS1, to implement wrap-around. This wrap-around implementation is apostrophied because it has been replaced by

another wrap around implementation.

```
2360  'IF XIP1N%<0 THEN 5180 ELSE 5220
2380  '           XIP1N%=XIP1N%+KS1%: GOTO 5300      'WRAP ARC
2400  'IF XIP1N%>KS1% THEN 5260 ELSE 5300
2420  '           XIP1N%=XIP1N%-KS1%
```

The following code calculates the YIP1 initial point Y- coordinate for field-1 as a function of the viewport centerpoint Y-position, YC1; the rotational Y-offset position, TY; and the geometric window parameters, KF5 and KF7.

```
2440  YIP1N%=YC1+TY+KF7-KF5
       'EQUIVALENT TO YPI1%=YC1-R1*8*SIN(A5%)/2
```

The following code provides an alternate implementation that detects the need for Y-axis wrap-around and controls Y-axis wrap-around. The IF THEN ELSE instruction test for the window YIP1N parameter exceeds the image memory boundaries and therefore determines if wrap-around is needed. If needed, the YIP1N parameter is updated by adding or subtracting the image memory Y- dimension, KS2, to implement wrap-around. This wrap-around implementation is apostrophied because it has been replaced by another wrap-around implementation.

```
2460  'IF YIP1N%<0 THEN 5380 ELSE 5420
2480  '           YIP1N%=YIP1N%+KS2%: GOTO 5500      'WRAP-AROUND
2500  'IF YIP1N%>KS2% THEN 5460 ELSE 5500
2520  '           YIP1N%=YIP1N%-KS2%
```

The following code was used to experiment with buffer memory and anti-aliasing but is no longer used and is therefore apostrophied-out.

```
2540  '           XIP1N%=XIP1N%-XRN%  'BUFFER MEMORY WITH ANTI-ALIASING
```

2560 YIP1N% = YIP1N% - YRN%

The following code calculates the XIP2 and YIP2 initial point coordinates for field-2 as a function of the field-1 initial point coordinates and the row slopes. The scaled slope parameters, XRN and YRN, are added to the field-1 initial positions, XIP1 and YIP1, to obtain the field-2 initial positions, XIP2 and YIP2.

2580 XIP2N% = XIP1N% + XRN% \* (.015625)

'XIP1-SF=8, XR%-SF=256; 1/2\*(8/256)=1/64

2600 YIP2N% = YIP1N% + YRN% \* (.015625)

The following code provides inputting and processing of joystick information. The operation of the ADCs is initiated with the frame sync signal going low, as discussed with reference to Figs 6U and 6V herein. The ADCs require less than 100-microseconds to perform a conversion. Hence, the ADCs are sampled more than 100-microseconds after the frame sync signal went low. This time delay is provided by the output operations; interpolation; and XIP1, XIP2, YIP1, and YIP2 processing preceding joystick processing in this time-slice.

2620 'JOYSTICK PROCESSOR; MUST OCCUR MORE THAN 100-MICROSECONDS AFTER FRAME SYNC GOES LOW

The following code outputs the joystick select address to select the ADC to be multiplexed onto the joystick data bus with the OUT 236 instruction, as discussed with reference to Figs 6U and 6V herein and inputs the joystick parameter with the INP (237) instruction. All four ADCs convert their respective parameters simultaneously and consequentially can be scanned in

sequence one after the other to access all four joystick parameters in the same time-slice, as discussed with reference to Figs 6U and 6V herein.

```
2640 OUT 236,66: JSXV%=INP (237): OUT 236,68:  
JSA%=INP (237)
```

```
2660 OUT 236,70: JSYV%=INP (237): OUT 236,64: JSS%=INP  
(237): OUT 236,80
```

The following code biases the acquired joystick parameters. The BX5, BY5, BS5, and BA5 numbers are the bias calibration numbers measured when the keyboard "J" key is depressed, performed when the joysticks are in the centered positions. Therefore, these numbers represent the values of the joysticks at the null positions. Subtracting of these bias calibration numbers from the measured joystick numbers provides the biased and calibrated values of the joysticks.

```
2680 JSXB%=JSXV%-BX5%: JSYB%=JSYV%-BY5%: JSSB%=JSS%-  
BS5%: JSAB%=JSA%-BA5%
```

The following code checks if the rotation joystick command is outside of the deadband; where the deadband is defined with DB1 and typically has a value of 20. If the joystick number is within the deadband, the joystick command is ignored by branching over joystick rotation command processing with the IF THEN instruction. If the joystick number is outside of the deadband, the joystick number is processed.

```
2700 IF ABS(JSAB%)<DB1% THEN 2800 'CHECK IF IN DEADBAND
```

The following code re-inserts the deadband that was subtracted-out to provide improved resolution, executing the command including the deadband component. If the rotation command is positive, then the deadband parameter is put back in by subtraction. If the rotation command is negative, then the deadband parameter is put back in by addition.

```
2720 IF JSAB%<0 THEN 2740 ELSE 2760 'BIAS OUT DEADBAND  
2740 JSAB%=JSAB%+DB1%: GOTO 2780  
2760 JSAB%=JSAB%-DB1%
```

The following code updates the angular position AR with the joystick rotational command, JSAB. The joystick command is processed to provide square-law scaling to introduce a non-linearity to the command and hence to increase the dynamic range. The linear command is converted to a non-linear square-law command by multiplying the joystick command by its absolute magnitude, thereby obtaining a number equal to the square of the command and preserving the sign of the command.

```
2780 AR=AR+JSAB%*ABS(JSAB%)*(.00003)  
'SQUARE LAW JOY STICK SCALING
```

The following code returns to the executive routine after the above time-slice processing has been completed.

```
2800 RETURN
```

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters. The PR9\$ parameter branches over the interpolation subroutine calls if interpolation once each frame was selected instead of interpolation once each field.

```
2820 *****
```

2840 IF PR9\$="N" GOTO 2880  
2860 GOSUB 5220: GOSUB 5840 'INTERPOLATION

The following code calls the keyboard routine to check for keyboard commands to exit the iterative processing or to recalibrate the joysticks.

2880 GOSUB 4980 'KEYBOARD INPUT ROUTINE

The following code conditionally clears the sum of the remainders in conjunction with generation the new interpolation iteration under control of the PR10\$ operator selected condition. This is for an experimental investigation to confirm the effects of clearing and of not clearing the remainders after each interpolation iteration.

2900 IF PR10\$="N" THEN 2960  
2920 SDXIP1R%=0: SDYIP1R%=0: SDXIP2R%=0: SDYIP2R%=0  
2940 SDXPR%=0: SDYPR%=0: SDXRR%=0: SDYRR%=0

F  
The following code calculates parameters for the next interpolative iteration. It calculates the delta initial position parameters and the delta slope parameters (beginning with D) for interpolation by subtracting each prior parameter (ending in P) from the related next parameter (ending in N). The delta initial position parameters represent the change in the initial position parameters over the eight field period of an interpolation iteration. The delta slope parameters represent the change in the slope parameters over the eight field period of an interpolation iteration.

2960 DXIP1%=(XIP1N%-XIP1P%): DYIP1%=(YIP1N%-YIP1P%)  
2980 DXIP2%=(XIP2N%-XIP2P%): DYIP2%=(YIP2N%-YIP2P%)

```
3000 DYP%=(YPN%-YPP%):      DXP%=(XPN%-XPP%)  
3020 DYR%=(YRN%-YRP%):      DXR%=(XRN%-XRP%)
```

The following code calculates parameters for the next interpolative iteration. It calculates the remainders (ending in R) for the initial position delta parameters (begining with D) and for the slope parameters. This processing works for both positive and negative delta numbers. The remainders are calculated by masking the least significant remainder bits of the delta parameters with the AND instruction. The mask is the number 3 for once per field interpolative updates and is the number 7 for once per frame updates, as controlled by the PR9\$ condition; masking off the subpixel resolution bits of each delta parameter as the remainder.

```
3040 'REMAINDER PROCESSING WORKS FOR + AND - DELTA  
      NUMBERS  
3060 IF PR9$="Y" GOTO 3180  
3080 DXIP1R%=DXIP1% AND 3:   DYIP1R%=DYIP1% AND 3  
3100 DXIP2R%=DXIP2% AND 3:   DYIP2R%=DYIP2% AND 3  
3120 DXPR%=DXP% AND 3:     DYPR%=DYP% AND 3  
3140 DXRR%=DXR% AND 3:     DYRR%=DYR% AND 3  
3160 GOTO 3260  
3180 DXIP1R%=DXIP1% AND 7:  DYIP1R%=DYIP1% AND 7  
3200 DXIP2R%=DXIP2% AND 7:  DYIP2R%=DYIP2% AND 7  
3220 DXPR%=DXP% AND 7:    DYPR%=DYP% AND 7  
3240 DXRR%=DXR% AND 7:    DYRR%=DYR% AND 7
```

The following code calculates parameters for the next interpolative iteration. It calculates the delta update per interpolation update. For an eight time-slice interpolation

iteration, each delta parameter is divided by eight so that the total update over eight update time-slices will equal the total delta for that interpolative iteration. For a four time-slice interpolation iteration, each delta parameter is divided by four so that the total update over four update time-slices will equal the total delta for that interpolative iteration. The time-slice parameter TS1 has previously been calculated at lines 820 to 860 to provide the appropriate factor of 1/8 or 1/4 to obtain the delta parameter per time-slice period.

```
3260 DXIP1% =DXIP1%*TS1:      DYIP1% =DYIP1%*TS1  
3280 DXIP2% =DXIP2%*TS1:      DYIP2% =DYIP2%*TS1  
3300 DXP% =DXP%*TS1:          DYP% =DYP%*TS1  
3320 DXR% =DXR%*TS1:          DYR% =DYR%*TS1
```

The following code returns to the executive routine after the above time-slice processing has been completed.

```
3340 RETURN
```

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters.

```
3360 *****  
3380 GOSUB 5220: GOSUB 5840  'INTERPOLATE
```

The following code calculates the sine of the rotation angle AR.

```
3400 KSAR=SIN(AR)
```

The following code returns to the executive routine after the above time-slice processing has been completed.

```
3420 RETURN
```

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters. The PR9\$ parameter branches over the interpolation subroutine calls if interpolation once each frame was selected instead of interpolation once each field.

```
3440 *****  
3460 IF PR9$="N" GOTO 3500  
3480          GOSUB 5220: GOSUB 5840 'INTERPOLATE
```

The following code calculates the cosine of the rotation angle AR.

```
3500 KCAR=COS(AR)
```

The following code returns to the executive routine after the above time-slice processing has been completed.

```
3520 RETURN
```

The following code resets the time-slice counter so that the ON GOTO instruction at line 2260 can wrap-around from the time-slice at line 3540 to the time-slice at 4000.

```
3540 *****  
3560 F6%=0           'RESET TIME SLICE COUNTER
```

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters.

```
580 GOSUB 5220: GOSUB 5840 'INTERPOLATE, FIRST  
OPERATION IN PRIOR TIME SLICE
```

The following code calculates the resolved joystick translation commands. The joystick commands are implemented to be in viewport coordinates. Consequently, rotation of the image rotates the image in the image memory relative to the viewport,

where the joystick translation axis, which are image memory coordinate related, also rotate. In order to reference the translation axes to the viewport coordinates, the joystick translation commands are resolved from image memory coordinates into viewport coordinates by summing the resolved components from the image memory coordinates to obtain the translation commands in viewport coordinates.

3600 JSYK% = JSYB% \* KSAR: JSXK% = JSXB% \* KSAR

3620 JSXB% = JSXB% \* KCAR + JSYK%: JSYB% = JSYB% \* KCAR - JSXK%

The following annotation makes note that the viewport position parameters, XC1 and YC1, should be updated prior to calculating the XIP and YIP parameters so that the XIP and YIP parameters are updated based upon the new XC1 and YC1 parameters.

3640 'UPDATE POSITION; PRECEDES XIP, YIP PROCESSING

The following code checks if the X-translational joystick command is outside of the deadband; where the deadband is defined with DB1 and typically has a value of 20. If the joystick number is within the deadband, the joystick command is ignored by branching over joystick rotation command processing with the IF THEN instruction. If the joystick number is outside of the deadband, the joystick number is processed.

3660 IF ABS(JSXB%) < DB1% THEN 3980

F The following code limits the X-translation to a <sup>convenient</sup> number, 30000 in this case, to form a hard-stop. However, wrap-around is being performed, as discussed herein, and hence the X-translation hard-stop is apostrophied-out.

3680 ' IF XC1 > 30000 AND JSXB% < 0 THEN

```
7420  'LIMIT X-TRANSLATION, SCROLLING  
3700  '           IF XC1<-30000 AND JSXB%>0 THEN  
7420
```

The following code implements X-translation wrap-around;  
causing the image processor to wrap-around the XC1 parameter at a  
~~convencient~~  
<sup>f</sup>  
<sup>convinient</sup> number, 30000 in this case, to form unlimited wrapped-  
around motion that folds back on itself as if image memory was  
formed as the inside of a sphere.

```
3720  IF XC1>30000 THEN 3740 ELSE 3760  
      'LIMIT X-TRANSLATION, WRAP-AROUND  
3740  XC1=XC1-32760: GOTO 3800  
      '(512*8)PIXELS*8 SUBPIXELS-8 SUBPIXELS  
3760          IF XC1<-30000 THEN 3780 ELSE 3800  
3780          XC1=XC1+32760
```

The following code re-inserts the deadband that was  
subtracted-out to provide improved resolution, executing command  
including the deadband component. If the X-translation command  
is positive, then the deadband parameter is put back in by  
subtraction. If the X-translation command is negative, then the  
deadband parameter is put back in by addition.

```
3800          IF JSXB%<0 THEN 3820 ELSE 3840  
          'BIAS OUT DEADBAND  
3820          JSXB%=JSXB%+DB1%: GOTO 3860  
3840          JSXB%=JSXB%-DB1%
```

The following code implements X-translation integer pixel  
processing. Fractional selection loops around this integer  
processing. The previous remainder DXR1 is updated DX1 with the  
next command from the X-joystick. The updated remainder DX1 is

FIXed to provide the integer part DXI1 of the new remainder. The updated remainder and the integer part thereof are subtracted to provide the fractional part DXR1 of the updated remainder to be saved for the next iteration. The remainder is saved to preclude accumulation of error. Then, the X-translation position XC1 is updated with the integer portion of the new remainder to provide the new integer pixel position XC1.

```
3860      IF PR16$="F" THEN 3960
3880          DX1=DXR1+JSXB%*ABS(JSXB%)*(.075)
3900          DXI1=FIX(DX1)
3920          DXR1=DX1-DXI1
3940          XC1=XC1-DXI1: GOTO 3980
```

The following code updates the X-translational position XC1 with the joystick X-translational command, JSXB. The joystick command is processed to provide square-law scaling to introduce a non-linearity to the command and hence to increase the dynamic range. The linear command is converted to a non-linear square law command by multiplying the joystick command by its absolute magnitude, thereby obtaining a number equal to the square of the command and preserving the sign of the command.

```
3960          XC1=XC1-JSXB%*ABS(JSXB%)*(.075)
```

The following code returns to the executive routine after the above time-slice processing has been completed.

```
3980  RETURN
```

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters.

```
4000 *****  
4020 OUT 236,65 'OUTPUT START PULSE, INCREMENT  
TIME SLICE COUNTER
```

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters. The PR9\$ parameter branches over the interpolation subroutine calls if interpolation once each frame was selected instead of interpolation once each field.

```
4040 IF PR9$="N" GOTO 4080  
4060 GOSUB 5220: GOSUB 5840 'INTERPOLATE
```

The following code checks if the Y-translational joystick command is outside of the deadband; where the deadband is defined with DB1 and typically has a value of 20. If the joystick number is within the deadband, the joystick command is ignored by branching over joystick rotation command processing with the IF THEN instruction. If the joystick number is outside of the deadband, the joystick number is processed.

```
4080 IF ABS(JSYB%)<DB1% THEN 4400
```

F The following code limits the Y-translation to a <sup>convenient</sup> ~~convenient~~ number, 30000 in this case, to form a hard-stop. However, wrap-around is being performed, as discussed herein, and hence the Y-translation hard-stop is apostrophied-out.

```
4100 ' IF YC1>30000 AND JSYB%>0 THEN 7740  
'LIMIT Y-TRANSLATION, SCROLLING  
4120 'IF YC1<-30000 AND JSYB%<0 THEN 7740
```

F The following code implements Y-translation wrap-around; causing the image processor to wrap-around the YC1 parameter at a <sup>convenient</sup> ~~convenient~~ number, 30000 in this case, to form unlimited wrapped-

around motion that folds back on itself as if image memory was formed as the inside of a sphere.

```
4140           IF YC1>30000 THEN 4160 ELSE 4180
                  'LIMIT X-TRANSLATION, WRAP-AROUND
4160           YC1=YC1-32760: GOTO 4220
4180           IF YC1<-30000 THEN 4200 ELSE 4220
4200           YC1=YC1+32760
```

The following code re-inserts the deadband that was subtracted-out to provide improved resolution, executing the command including the deadband component. If the Y-translation command is positive, then the deadband parameter is put back in by subtraction. If the Y-translation command is negative, then the deadband parameter is put back in by addition.

```
4220   IF JSYB%<0 THEN 4240 ELSE 4260  'BIAS OUT DEADBAND
4240           JSYB%=JSYB%+DB1%: GOTO 4280
4260           JSYB%=JSYB%-DB1%
```

The following code implements Y-translation integer pixel processing. Fractional selection loops around this integer processing. The previous remainder DYR1 is updated DY1 with the next command from the Y-joystick. The updated remainder DY1 is FIXed to provide the integer part DYI1 of the new remainder. The updated remainder and the integer part thereof are subtracted to provide the fractional part DYR1 of the updated remainder to be saved for the next iteration. The remainder is saved to preclude accumulation of error. Then, the Y-translation position YC1 is updated with the integer portion of the new remainder to provide the new integer pixel position YC1.

```
4280      IF PR16$="F" THEN 4380
4300          DY1=DYR1+JSYB%*ABS(JSYB%)*(.075)
4320          DYI1=FIX(DY1)
4340          DYR1=DY1-DYI1
4360          YC1=YC1-DYI1: GOTO 4400
```

The following code updates the Y-translational position YC1 with the joystick Y-translational command, JSYB. The joystick command is processed to provide square-law scaling to introduce a non-linearity to the command and hence to increase the dynamic range. The linear command is converted to a non-linear square-law command by multiplying the joystick command by its absolute magnitude, thereby obtaining a number equal to the square of the command and preserving the sign of the command.

```
4380          YC1=YC1+JSYB%*ABS(JSYB%)*(.075)
```

The following code checks if the scale factor joystick command is outside of the deadband; where the deadband is defined with DB1 and typically has a value of 20. If the joystick number is within the deadband, the joystick command is ignored by branching over joystick rotation command processing with the IF THEN instruction. If the joystick number is outside of the deadband, the joystick number is processed.

```
4400  IF ABS(JSSB%)<DB1% THEN 4620
```

The following code re-inserts the deadband that was subtracted-out to provide improved resolution, executing the command including the deadband. If the scale factor command is positive, then the deadband parameter is put back in by subtraction. If the scale factor command is negative, then the deadband parameter is put back in by addition.

```
4420 IF JSSB%<0 THEN 4440 ELSE 4460
4440      JSSB%=JSSB%+DB1%: GOTO 4480  'BIAS OUT DEADBAND
4460      JSSB%=JSSB%-DB1%
```

The following code calculates the scale factor update parameter DS11 with the joystick scale factor command, JSSB. The joystick command is processed to provide square-law scaling to introduce a non-linearity to the command and hence to increase the dynamic range. The linear command is converted to a non-linear square-law command by multiplying the joystick command by its absolute magnitude, thereby obtaining a number equal to the square of the command and preserving the sign of the command. The scale factor square-law command is scaled with a small number, 0.00001, and added to unity to provide an offset binary scale factor. Unity scale factor provides no change in size. Hence, adding the scaled offset binary scale factor command to unity provides a slightly greater than unity scale factor or a slightly less than unity scale factor for positive or negative zoom.

```
4480      DS11=1+JSSB%*ABS(JSSB%)*(.00001)
                  'OFFSET BINARY TO SIGN BINARY, SCALE,
                  BIAS ABOUT UNITY
```

The following code limits the amount of expansion and compression to the values selected by the operator. It tests the XS-parameter and the YS-parameter to confirm that they are within the selected positive and negative limits. If within these limits, scaling processing is performed. If not within these limits, scaling processing is looped over.

```
4500      IF DS11<1 AND XS<PR12 OR DS11>1 AND YS<PR12
```

THEN 4620

520 IF DS11>1 AND XS>PR13 OR DS11>1 AND YS>PR13 THEN  
4620

4540 XS=XS\*DS11: YS=YS\*DS11

The following code updates scale factor parameters as a function of whether the old processing or the new processing was selected.

4560 IF PR32\$="Y" GOTO 4600

4580 YSS=YS\*(3.90625E-03): XSSV=XS\*XSVR: GOTO 4620

4600 KB1=KB1\*DS11: KB2=KB2\*DS11

The following code returns to the executive routine after the above time-slice processing has been completed.

4620 RETURN

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters.

4640 \*\*\*\*\*

4660 GOSUB 5220: GOSUB 5840 'INTERPO

#### CALCULATIONS PERFORMED IN PRIOR TIME-SLICE

The following code calculates the KB4, KB5, KB6, and KB7 parameters as a function of whether the old processing or the new processing was selected.

4680 IF PR32\$="Y" GOTO 4740

4700 KF6=KB2\*KCAR\*XSSV: KF7=KB1\*KCAR\*YSS

4720 KF4=KB1\*KSAR\*XSSV: KF5=KB2\*KSAR\*YSS: GOTO 4780

4740 KF6=KB2\*KCAR: KF7=KB1\*KCAR

4760 KF4=KB1\*KSAR: KF5=KB2\*KSAR

The following code returns to the executive routine after the above time-slice processing has been completed.

4780 RETURN

The following code, at the start of the time-slice processing, calls the interpolation routines to generate the next interpolated output parameters. The PR9\$ parameter branches over the interpolation subroutine calls if interpolation once each frame was selected instead of interpolation once each field.

4800 \*\*\*\*\*

4820 IF PR9\$="N" GOTO 4860

4840 GOSUB 5220: GOSUB 5840

'INTERPOLATE

The following code generates the slope~~s~~ parameters as trigonometric functions of the scale factor parameters XS and YS. The CINT instruction is conditionally used as a function of operator selection, as discussed above.

4860 IF PR11\$="N" THEN 4920

4880 YPN% = CINT(KSAR\*XS): XRN% = CINT(2\*KSAR\*YS)

'CALCULATE SLOPES

4900 YRN% = -CINT(2\*KCAR\*YS): XPN% = CINT(KCAR\*XS): GOTO 4960

4920 YPN% = KSAR\*XS: XRN% = 2\*KSAR\*YS

'CALCULATE SLOPES

4940 YRN% = -2\*KCAR\*YS: XPN% = KCAR\*XS

The following code returns to the executive routine after the above time-slice processing has been completed.

4960 RETURN

The following code interrogates the keyboard, where the port is defined as a function of the system selected. The J-key selects joystick recalibration. The DELETE-key selects return to the operating system. The ESCAPE-key selects return to the DIS.ASC menu.

```
4980 *****  
5000 IF K2$="C" THEN 5040           'KEYBOARD  
      COMMANDS  
5020 A7%=INP (1): A8%=INP (0): GOTO 5060  
5040 A7%=INP (93): A8%=INP (92)  
5060 A7%=A7% AND 2: IF A7%=0 THEN 5200   'DATA  
      READY TEST  
5080 A8%=A8% AND 127          'MASK PARITY BIT  
5100 A9%=A8% XOR 27: IF A9%=0 THEN 240   'ESCAPE  
      TO MENU  
5120 A9%=A8% XOR 74: IF A9%=0 THEN 5140 ELSE 5160  
      '"J" TO RECALIBRATE JOYSTICKS  
5140     BS5%=JSS%: BX5%=JSXV%: BY5%=JSYV%: BA5%=JSA%:  
      GOTO 5200  
5160 A9%=A8% XOR 127: IF A9%=0 THEN 5180 ELSE 5200  
      'DELETE TO CP/M  
5180     SYSTEM  
5200     RETURN
```

The following code provides the interpolation processing per time-slice. The interpolation processing was divided <sup>into</sup> ~~in~~ two subroutines so that they could be partitioned between time-slices to optimize performance by minimizing the number of time-slices. However, both subroutines are called from the same time slice in

this program because the performance is appropriate.

```
5220  ****
5240  'INTERPOLATION SUBROUTINE-1
5260  'UPDATE INITIAL POSITIONS AND SLOPES
```

The following code accumulates the sum of the remainders by adding the new remainder to the prior sum of the remainders to obtain the new sum of the remainders for each position and slope parameter that is to be output.

```
5280  'REMAINDER PROCESSING
5300  SDXIP1R%=SDXIP1R%+DXIP1R%: SDYIP1R%=SDYIP1R%+DYIP1R%
5320  SDXIP2R%=SDXIP2R%+DXIP2R%: SDYIP2R%=SDYIP2R%+DYIP2R%
5340  SDXPR%=SDXPR%+DXPR%:           SDYPR%=SDYPR%+DYPR%
5360  SDXRR%=SDXRR%+DXRR%:           SDYRR%=SDYRR%+DYRR%
```

The following code calculates the new parameters by adding the delta parameter to the prior parameter to get the new parameter for the current time-slice for each position and slope parameter that is to be output.

```
5380  'INTERPOLATION DELTA UPDATES
5400  XIP1P%=XIP1P%+DXIP1%: YIP1P%=YIP1P%+DYIP1%
5420  XIP2P%=XIP2P%+DXIP2%: YIP2P%=YIP2P%+DYIP2%
5440  XPP%=XPP%+DXP%:           YPP%=YPP%+DYP%
5460  XRP%=XRP%+DXR%:           YRP%=YRP%+DYR%
```

The following processing tests each sum of the remainder parameters to determine if it has an integer portion, being greater than the eighth pixel resolution of the processing. If not, operation proceeds to the next operation. If yes, the sum of the product parameter is decremented by one pixel to eighth pixel

resolution and the related initial position or slope parameter is incremented by one pixel to integer pixel resolution.

```
5480 IF SDXIP1R%<8 THEN 5520
      SDXIP1R%=SDXIP1R%-8: XIP1P%=XIP1P%+1
5520 IF SDYIP1R%<8 THEN 5560
      SDYIP1R%=SDYIP1R%-8: YIP1P%=YIP1P%+1
5560 IF SDXIP2R%<8 THEN 5600
      SDXIP2R%=SDXIP2R%-8: XIP2P%=XIP2P%+1
5600 IF SDYIP2R%<8 THEN 5640
      SDYIP2R%=SDYIP2R%-8: YIP2P%=YIP2P%+1
5640 IF SDXPR%<8 THEN 5680
      SDXPR%=SDXPR%-8: XPP%=XPP%+1
5680 IF SDYPR%<8 THEN 5720
      SDYPR%=SDYPR%-8: YPP%=YPP%+1
5720 IF SDXRR%<8 THEN 5760
      SDXRR%=SDXRR%-8: XRP%=XRP%+1
5760 IF SDYRR%<8 THEN 5800
      SDYRR%=SDYRR%-8: YRP%=YRP%+1
5800 '
```

The following code returns to the time-slice routine that called the interpolation subroutine.

```
5820 RETURN
```

The following code performs further interpolation processing.

```
5840 ****
5850 'INTERPOLATION SUBROUTINE-2
```

The following code formats the output parameters in response to the updated position and slope parameters. The formatting generates the LSH parameters by ANDing each parameter with an LSH-mask to preserve the 6-LSBs and to mask-out any MSBs. The formatting generates the MSH parameters by scaling each parameter with 1/64 or 0.015625 and taking the integer portion thereof with the integer symbol "%" to place the MSBs into the LSB position of the output parameter. Integer or non-integer (fractional) parameters are generated under control of the PR21\$ selection parameter. If non-integer parameters are selected, the 3-LSBs are maintained with a mask of 64. If integer parameters are selected, the 3-LSBs are set to zero with a mask of 56 and the 3-LSBs of the MSH are masked with a mask of 60.

```
5860  'FORMAT INITIAL POINT OUTPUTS
      IF PR21$="N" THEN 6020
      CA1%=YIP1P%*(.015625): CB1%=YIP1P% AND 63:
      CC1%=XIP1P%*(.015625): CD1%=XIP1P% AND 63
      5920  CA2%=YIP2P%*(.015625): CB2%=YIP2P% AND 63:
      CC2%=XIP2P%*(.015625): CD2%=XIP2P% AND 63
      5940  'FORMAT SLOPE OUTPUTS
      5960  XPM%=XPP%*(.015625): XPL%=XPP% AND 63:
              YPM%=YPP%*(.015625): YPL%=YPP% AND 63
      5980  XRM%=XRP%*(.015625): XRL%=XRP% AND 63:
              YRM%=YRP%*(.015625): YRL%=YRP% AND 63
      6000  GOTO 6140
      6020  'FORMAT IP AND SLOPE AND TRUNCATE TO PIXEL
              RESOLUTION
```

```
6040 CA1% = YIP1P% * (.015625) : CB1% = YIP1P% AND 56:  
CC1% = XIP1P% * (.015625) : CD1% = XIP1P% AND 56  
6060 CA2% = YIP2P% * (.015625) : CB2% = YIP2P% AND 56:  
CC2% = XIP2P% * (.015625) : CD2% = XIP2P% AND 56  
6080 XPM% = XPP% * (.015625) AND 60: XPL% = XPP% AND 0:  
YPM% = YPP% * (.015625) AND 60: YPL% = YPP% AND 0  
6100 XRM% = XRP% * (.015625) AND 60: XRL% = XRP% AND 0:  
YRM% = YRP% * (.015625) AND 60: YRL% = YRP% AND 0  
6120 '
```

The following code returns to the time-slice routine that called the interpolation subroutine.

```
6140 RETURN
```

The following code is an END statement placed at the end of the listing.

```
6160 *****  
6180 END
```

### Circuit Boards

The experimental system is implemented with wire wrap circuit boards consisting of 2-Memory Boards (BM1 and BM2), 1-Logic Board (BL1), 1-Buffer Board (BB1), and 1-Rear End Board (BR1). Each board is constructed with a Vector board, manufactured by Vector Electronic Company of Sylmar CA, having 1/10th inch hole spacings on a 17-inch by 8 1/2-inch board. Wire wrap DIP sockets and cable connectors are inserted into the Vector board and interconnected with wire wrap interconnections. Information on the DIPs plugged-in to the DIP sockets is provided for selected boards in the printout of the TABLE OF DIP LAYOUT ON BOARDS included herewith. Information on the cable connectors is provided for each cable in the printout of the CABLE CONNECTION TABLE included herewith.

DIP assignments are provided for selected boards in the TABLE OF DIP LAYOUT ON BOARDS included herewith, for each board. DIPs are arranged on the boards as rows identified with alphabetical symbols; i.e., A to E; and as columns identified with numerical symbols; i.e., 1 to 23. Each DIP position on a board is identified with a U symbol followed by the column and row symbols (i.e., U3A).

Logical schematic diagrams showing implementation of the experimental system are provided herewith, such as shown in Fig 6. These logical diagrams show standard commercially available integrated circuits; such as TTL series 7400 ICs, Mitsubishi M58725 RAMs, and 8216 bi-directional bus drivers, and ~~Sygnetics~~ 8T97 buffers. Specifications for these integrated circuits are available in catalogs and specification sheets from the

abovementioned manufacturers and are well known in the art.

The schematic diagrams show the logical function in symbolic form, identify the type of IC, identify the DIP numbers and pin numbers, and show wiring interconnections between DIP and pin numbers. Device types are often shortened, such as shortening the name 74LS02 to LS02. DIP assignments are identified with U numbers, such as U20C representing the DIP at row-20 column-C on the subject circuit board. For example, a NOR-gate is shown at the top portion of Fig 6B identified with the designation LS02 as being a 74LS02 quad NOR-gate integrated circuit, identified with the designation U21C as being located on the BL1 circuit board at row-C column-21, and having 2-input signals on pin-11 and pin-12 of the DIP and one output signal on pin-13 of the DIP. The input to pin-11 is shown connected to the output of DIP U22B pin-3, the input to pin-12 is shown connected to the output of DIP U20B pin-12, and the output from pin-13 is shown connected to DIP U7D pin-9. For convenience of documentation, interconnections may be designated by the DIP identification number and pin number separated by a dash; i.e., U21C-13 representing pin-13 of DIP U21C. For convenience of discussion, logical circuits may be designated by the DIP identification number and the output pin number separated by a dash; i.e., U21C-13 representing pin-13 of DIP U21C.

### Cable List

A cable list is provided in the CABLE CONNECTION TABLE included herewith. This cable list identifies the cables between the various Vector boards and between the Vector boards and the supervisory processor. Each cable between display processor boards is implemented with a 50-pin ribbon cable having odd pins connected to ground for signal isolation. Each cable between the Vector boards and the supervisory processor is implemented with an RS-232 type 25-pin ribbon cable, consistent with the signal representations for the Compupro Interfacer-II board. The cable list identifies the pin associated with a signal, a symbol associated with the signal, a description of the signal, a representative source of the signal and a representative destination of the signal.

## S-100 Bus System

The experimental system has been implemented with an S-100 bus based system performing supervisory processor functions in conjunction with the novel software and hardware, as discussed herein. Two S-100 bus based systems have been configured, the Camille system and the Murphy system. The configuration of the Camille system will be discussed in detail hereinafter. The Camille system comprises a computer, a pair of floppy disk drives, a terminal, and printers as discussed below.

The floppy disk drives are implemented with a pair of 8-inch disk drives in an enclosure and operating in conjunction with a DMA controller in the computer. The disk drives are manufactured by Siemens as the FDD 100-8; the drive enclosure is manufactured by International Instrumentation, Incorporated; and the DMA Controller is manufactured by CompuPro as the Disk 1 DMA Controller; all described in detail in the referenced manuals.

The terminal is manufactured by Applied Digital Data Systems, Inc. (ADDS) as the Model Viewpoint/3A Plus; described in detail in the referenced manual.

The printers include a dot matrix printer manufactured by Star Micronics, Inc. as the Gemini-10, a dot matrix printer manufactured by Epson as the FX-100, and a daisywheel printer manufactured by Smith-Corona as the TP-I; all described in detail in the referenced manuals.

The computer is implemented with a cabinet manufactured by Fulcrum Computer Products as the I8080 Microcomputer System Cabinet and having a backplane S-100 board manufactured by CPA which is described in detail in the referenced CPA manual.

The computer is implemented with various S-100 boards manufactured by CompuPro including the 8085-8088 CPU board, RAM 16 and RAM 17 memory boards, a System Support board, and a pair of Interfacer 2 boards. One Interfacer 2 board is used to interface to the terminal and printers. The other Interfacer 2 board provides the 3-channel parallel interface to the control logic. These boards are described in detail in the referenced manuals.

The joysticks are implemented with the Computer Compatible Joystick; described in the referenced applications notes.

The operating system is CP/M 80, which is described in detail in the referenced documents.

The applications program runs under Basic, which is described in detail in the referenced documents.

The following documents provide supplemental data on the computer system and are herein incorporated by reference.

1. Technical Manual, Siemens, OEM Floppy Disk Drive FDD 100-8, Technical Manual, Model 100-80.
2. Manual, International Instrumentation, Incorporated, Universal Disk Enclosures, General Information/Pricing, 1982.
3. User Manual, CompuPro Division of Godbout Electronics, Disk 1 Arbitrated 24 Bit DMA Floppy Disk Controller, 1981.
4. User's Manual, Applied Digital Data Systems, Inc., Viewpoint/3A Plus, 518-31100. print. r
5. Operation Manual, Star Micronics, Inc., Gemini-10.
6. Operation Manual, Epson, FX Printer, 1983.
7. Operator's Manual, Smith-Corona, TP-I.
8. Functional Description, CP-A, Revision 1.
9. Technical Manual, CompuPro Division of Godbout Electronics, 8085/8088 CPU Dual CPU, 2/83.
10. Technical Manual, CompuPro Division of Godbout Electronics, RAM 16 Static Memory, 4/82.
11. Technical Manual, CompuPro Division of Godbout Electronics, RAM 17 64K Static Memory, 9/82.
12. User's Manual, CompuPro Division of Godbout Electronics, System Support 1, 8/81.
13. Technical Manual, CompuPro Division of Godbout Electronics, Interfacer 2, 4/82.
14. The CP/M Handbook with MP/M, by Rodnay Zaks, published by Sybex, 1980.
15. CP/M Primer, by Stephen Murtha and Mitchell Waite, published by Howard W. Sams & Co., Inc., 1980.

16. An Introduction to CP/M Features and Facilities,  
published by Digital Research, January 1978.
17. Microsoft Basic Reference Book, published by Microsoft,  
1979.
18. Microsoft Basic Compiler Documentation, published by  
Microsoft.
19. The Basic Handbook (2nd Edition), by David Lien,  
published by Compusoft Publishing, 1981.
20. Microsoft Basic (2nd Edition), by Ken Knecht, published  
by Dilithium Press, 1983.
21. Basic Basic (2nd Edition), by James Coan, published by  
Hayden Book Company, Inc., 1978.
22. Computer Compatible Joystick Instruction, applicable to:  
Apple-II.

## LOGIC BOARD

### Control Logic

Various control arrangements can be provided for controlling operation. For example; counter, ROM, and logical control arrangements of synchronous or asynchronous design can be used. A gated clock arrangement has been implemented for control, which is illustrative of other forms of gated clock control logic and other non-gated clock control logic implementations. This gated clock control arrangement will now be discussed with references to Figs 6B to 6D.

The gated clock control logic shown in Figs 6B to 6D controls the clock pulses to various logic devices; such as address generators, memories, and other devices; to clock the various operations associated therewith. For example, the write strobe W-bar to the memories is generated with U22C-11 and clocks to various registers are gated with circuits U21C, U20C, U19C, U19D, U18C, U9A, and U10A (Fig 6B). Several different types of clocks are generated at different times and are controlled to be non-conflicting with other clocks. Load clocks are generated under computer control to load computer generated parameters into various registers. Address generator clocks can be generated under control of external sync pulses. High speed clocks can be generated under control of various signals to generate addresses that are not in contention with other signals generating clocks and are not generated at times that such addresses are not needed.

The logic composed of U18D-6, U22B-3, U21E-6, U19B, U20B, and U21D-6 (Fig 6B) is controlled by the computer for generating strobe signals and control signals for loading computer information into the address generators and for disabling operations during loading of computer information. Logic gates U20C and U21C (Fig 6B) gate the computer strobe to load delta registers. Logic gates U19C, U19D, and U18C (Fig 6B) gate various clock signals to load and update R-registers. Logic gates U9A and U10A (Fig 6B) gate various clock signals to load and update P-registers. Flip-flops U22E and associated logic U20D-4, U22A-4, U18D-8, U21E-4, U15A-3, U22C-3, U20D-13, U18D-2, U21E-10, U18E-6, U20D-1, U21E-12, U19D-4, U18E-3, U21D-8, U20E-6, U17A-8, U17A-11, U22A-2, U13A-8, U21E-2, U20E-3, and other related elements (Fig 6D) synchronize and process the sync signals CFS and CLS to generate clock and control signals for processor operation. Logic gates and flip-flops U12A, U21D-8, U18E-3, U14A-2, U14A-4, U16A-3, U17A-3, U16A-6, U17A-6, U13A-6, U23C-10, U14A-6, U20E-11, U21B-5, U15A-11, U21B-2, U21E-8, and related gates (Fig 6C) control the gated clock pulses for address generation operations.

The computer interface signals are defined in the tables of computer interface signals; PORT-A, PORT-B, AND PORT-C. Port-A input and output signals are control signals. Port-B output signals are address and data signals to load into the selected destination. Port-C output signals are destination select signals.

Computer load logic will now be discussed with reference to Fig 6B. Computer control signals DOA6 and DOA7 control loading of initial conditions. When DOA6 is 1-set, run operation is commanded and loading of initial conditions from the computer with the load strobe DOA7 is disabled with gate U22B-3. When DOA6 is 0-set, run operation is disabled and load operation is enabled through inverter U18D-6 by enabling gate U22B-3 to pass an inverted computer strobe DOA7 as signal U22B-3. This gated strobe is used to clock the selected register, steered with the register address decoders U19B and U20B to gates U21C, U20C, U19C, and U19D. The inverted DOA6 signal U18D-6 is inverted with U21E-6 to generate the DIEN-bar signal for memory read and write operations. When DOA6 is high, defining the run mode; DIEN-bar is high establishing the memory read mode. When DOA6 is low, defining the load mode; DIEN-bar is low establishing the memory write mode. DIEN-bar control operations are discussed in detail with reference to Figs 6E to 6N for the memory logic.

Write signals U22C-6 and U22C-8 control writing into a peripheral RAM by enabling write pulses W2 and W3 when addressed through U19B-10 and U19B-9 respectively.

Gate U22C-11 is an OR-gate that is enabled with the address of the memory U19B-11 to load data into the memory for steering of the computer strobe U22B-3 to generate a write strobe to load the computer generated parameter into memory.

Gates U20C and U21C are NOR-gates that are enabled with the destination register address signals from U19B and U20B to select the delta register to be loaded by steering of the computer strobe U22B-3 to clock the appropriate delta register to load the

computer generated parameter into the selected register.

A master clock, shown as CPE-bar (Fig 6D), is fanned-out, gated, and applied to the synchronous elements. The CPE-bar clock is derived from a clock pulse generator and communicated to the logic board on cable C4-6. Alternately, clock CPE-bar can be generated with a self-contained clock generating operating asynchronously with reference to the external clock to permit optimization of clock periods for system operation. For example, the external clock may be constrained to a clock frequency consistent with the requirements of a sync generator for the CRT monitor, which may not be an optimum clock frequency for the address generators. Therefore, a separate clock can be provided having a clock frequency that is optimum for the address generators in place of the external clock.

Clock logic will now be discussed with reference to Fig 6D. The clock CPE-bar is logically processed to clock synchronous elements in the address generators at the same time. For convenience of definition, clocking occurs at the rising edge of the delayed clock pulse CPD, which is delayed by 2-gate propagation delays after the early clock CPE-bar and one inversion of the early clock CPE-bar. For example, CPE-bar propagates through inverter U21E-2 and non-inverting AND-gate U20E-3 to provide one inversion and 2-delays to generate delayed clock CPD prior to being used to synchronously clock register U22E. Similarly, clock CPE-bar is delayed by gating logic U12A-8 and U12A-6 (Fig 6C), providing one stage of delay and one stage of inversion, and by non-inverting gates U9A-6, U9A-8, U10A-6,

and U10A-8 (Fig 6B) to provide the 2-propagation delays and the single inversion from the early clock CPE-bar to the clock signals for registers U8D, U9D, U5D, U8E, U5E, and U9E.

In one display configuration, register U22E is used to synchronize operation of the logic with a frame sync signal CFS and a line sync signal CLS. A short synchronous pulse is generated in conjunction with the line sync signal CLS. CFS is synchronously clocked into flip-flop U22E-10 and CLS is synchronously clocked into flip-flop U22E-12 to latch these signals as CFSR1 and CLSR1 respectively synchronous with the address generator clock. Latched line sync signal CLSR1 is then latched in flip-flop U22E-15 one clock period later for a delayed line sync signal CLSR2. The delayed line sync signal CLSR2 U22E-15 is inverted with inverter U21E-10 and NANDed with the non-delayed line sync signal CLSR1 with NAND-gate U18E-6 to generate a short inverted pulse bracketing the first clock period of line sync signal CFSR1.

Clock signal CPE-bar (Fig 6D) is gated with NAND-gates U12A-8 and U12A-6 (Fig 6C) to generate a gated clock signal for address generation; which is performed with registers U8D, U9D, U5D, U8E, U9E, and U5E. Gating of the clocks to these registers with AND-gates U9A-6, U9A-8, U10A-6, and U10A-8 gates address generator operations.

Control logic for a display configuration will now be discussed with reference to Fig 6D. Signal U13A-8 is a clock gate control signal for gating the address generator clock, as described herein with reference to Fig 6C. This gate signal is comprised of three components; U17A-8, U17A-11, and U22A-4.

These components cause the clock to be generated at the appropriate time in conjunction with the display sync signals.

Gate U20D-4 ORs together the <sup>1,1,1</sup> field sync and frame sync signals to enable the address generator clock through U22A-4 and U13A-8 when neither a frame sync signal CFSR1 nor a line sync signal CLSR1 is present. Gate U20E-6 ANDs together the inverted frame sync signal CFSR1 through inverter U18D-8 and the line sync signal. Gate U17A-8 NANDs together U20E-6 and the delayed line sync signal CLSR4 to enable the address generator clock through U13A-8. Gate U17A-11 NANDs together U20E-6 and the undelayed inverted line sync signal CLSR1 through inverter U22A-2 to enable the address generator clock through U13A-8.

The ELS signal U15A-3 controls multiplexers U10D, U11D, U12D, U10E, U11E, and U12E (Fig 6D). During the appropriate portions of the load mode, the address generator P-registers are loaded from the address generator R-registers under control of the ELS signal. During other periods of time, the address generator P-registers are updated from the related delta registers under control of the ELS signal. The ELS signal is disabled by the sequential load control signal DOA5 inverted with U21E-4. This permits the P-registers to be updated from the delta registers to generate vectors into memory, such as for a display configuration. The ELS signal is strobed with a short pulse U19D-4 during the load mode DOA6 as controlled with U22C-3. During the load mode, the ELS signal is enabled with DOA6 enabling U22C-3 to pass the short pulse U22C-2. The short pulse is generated by the early line sync signal CLSR1 U22E-12 and the

thrice delayed line sync signal CLSR4 U22E-7 for a 3-clock period transfer pulse to transfer information from the R-registers to the P-registers. The 3-period pulse U19D-4 is generated when CLSR4 U22E-7 is low and when CLSR1 U22E-10 is high, as inverted with U21E-12 to define the period that the undelayed line sync signal CLSR1 has gone high and before the delayed line sync signal CLSR4 has gone high; indicative of the first 3-clock periods at the start of a line sync pulse.

The XA3 and XA3-bar signals are shown gated with U19A-1 to disable both the XA3-bar signal and hence memory board-1 and the XA3 signal and hence memory board-2 with gates U19C-13 and U19C-4 respectively. This provides for blanking of the display and clearing of the buffer memory by outputting zeros from the disabled memory board when either the frame sync signal CFSR1 or the sequential load signal DOA5 are true. U19C-1 disables the memory boards during sequential loading with DOA5 U19A-3 and during the vertical blanking period with the inverted CFSR1 signal from inverter U18D-8.

Registers U22E and U23C are used to latch signals. U23C-2 and U23C-5 latch signals C2-30 and C2-32 to provide latched signals C4-32 and C4-36 respectively. U23C-10 is used in the clock gating logic, as discussed with reference to Fig 6C. U22E-10 and U22E-12 latch signals CFS and CLS respectively. U22E-15, U22E-2, and U22E-7 provide 1-clock delay, 2-clock delays, and 3-clock delays respectively for the CLSR1 signal.

Gated clock operations will now be discussed with reference to Fig 6C. Gated clock signals U12A-8 and U12A-6 each gate early clock CPE-bar with DOA6 from U21E-6 so that address generation will only be performed when the run/load-bar signal DOA6 is high, indicative of run operations. Gated clock signals U12A-8 and U12A-6 also gate early clock CPE-bar with U13A-8, which is composed of 3-gating conditions; U17A-8, U17A-11, and U22A-4; discussed in greater detail with reference to Fig 6D. Gated clock signal U12A-8 also gates early clock CPE with U14A-6, which enables high clock rate memory scanout operations within a block. Gated clock signal U12A-6 also gates early clock CPE-bar with U21B-2, which enables low clock rate memory block re-addressing operations. Consequently, when memory operations are proceeding within a block of 64-pixels, the address generator clock is generated as shorter period clock signal U12A-8 and, when memory operations are making a transition between blocks and need additional clock time for re-addressing, the address generator clock is generated as longer period clock signal U12A-6.

Determination of whether memory scanout or re-addressing is being performed for the particular clock period is determined by detecting an overflow of an address generator, as indicative of re-addressing, or detecting of a non-overflow of all address generators, as indicative of scanout. Overflow for this condition is defined as an overflow for a positive delta condition and an underflow for a negative delta condition. Therefore, detection of a carry condition for a positive delta or detection of a non-carry condition for a negative delta represents an overflow condition for gating a clock. An overflow

condition on either the X-address generator or the Y- address generator causes a re-addressing condition.

As shown in Fig 6C, a re-addressing condition is detected with NAND-gate U13A-6 from any one or combination of the 4- conditions U16A-3, U17A-3, U16A-6, and U17A-6. U16A-3 compares the inverted overflow bit C1-bar U15E-9 of the Y-address generator with the non-inverted sign bit SN1 U14E-6 of the Y- delta register to detect a Y-negative overflow condition. U17A-3 compares the overflow bit C1 U15E-9 of the Y-address generator with the inverted sign bit SN1 U14E-6 of the Y-delta register to detect a Y-positive overflow condition. U16A-6 compares the inverted overflow bit C2-bar U15D-9 of the X-address generator with the non-inverted sign bit SN2 U14D-6 of the X-delta register to detect an X-negative overflow condition. U17A-6 compares the overflow bit C2 U15D-9 of the X-address generator with the inverted sign bit SN2 U14D-6 of the X-delta register to detect a X-positive overflow condition. An overflow signal U13A-6 is latched and delayed with flip-flop D23C-10 for enabling of the scanout clock U12A-8 for scanout, in the absence of an overflow condition. Flip-flop U23C-10 provides a one-clock period delay so that an extended re-addressing clock period occurs in the clock period following the overflow condition, which is the clock period during which the re-addressing is performed. Latched overflow signal U23C-10 is inverted with inverter U14A to form a non-overflow signal U14A-6 and used to enable the scanout clock U12A-8.

Latched overflow signal U23C-10 is also processed with flip-flops U21B-5 and U21B-2 to provide a triple clock period for an overflow. These flip-flops are clocked with the non-gated delayed clock pulse CPD U20E-3 to control non-gated clock period time delays. If clocked with the gated clock pulse, such as with U15A-8; then the gating clock logic could cause the clock signal to lock-up.

A triple clock period for re-addressing will now be discussed with reference to Fig 6C. Detection of an overflow condition U23C-10 with gate U20E-11 sets flip-flop U21B-5 on the first clock period and sets flip-flop U21B-2 on the second clock period, which adds 2-clock periods to the basic single clock period; yielding a triple clock period to facilitate re-addressing. At the completion of the third clock period, the 1-set signal U21B-2 enables a single clock signal U12A-6 and is inverted to a 0-set signal U21E-8 to reset U21B-5 through U20E-11 and to reset U21B-2 through U15A-11 on the next clock to clock flip-flops U21B-5 and U21B-2, respectively. This triple clock period logic is designed to operate for a single overflow condition surrounded by non-overflow conditions, or for two overflow conditions immediately following each other, and for many overflow conditions immediately following each other. For a single overflow condition; scanout clock U12A-8 has a series of single period clocks and has 3-clock periods missing that are coincident with overflow conditions and re-addressing clock U12A-6 has a single clock coincident with the overflow condition. For multiple sequential overflow conditions; scanout clock U12A-8 has a series of single period clocks with a series of triple clock

periods missing that are coincident with the multiple sequential overflow conditions and re-addressing clock U12A-6 has multiple sequential clocks each separated by 2-clock periods.

The scanout clock U12A-8 and the re-addressing clock U12A-6 are generated separately for gating purposes. They are ORed together with gate U21D-8 for loading the buffer memory with signal C3-22 and for clocking register U23C-9. This causes the pipeline from the memory output through the buffer memory to be clocked by an out-of-phase signal, yielding a 1.5 clock period propagation delay time for the memory. The design is carefully configured so that the pipeline propagation delay is greater than . the 0.5 clock periods and less than the 1.5 clock periods to facilitate proper clocking of the memory output signal into the buffer memory with a propagation delay that can approach the 1.5 clock period.

The clock signals to the address generation registers are implemented as the logical-OR of a plurality of different clock signals. In order to equalize clock delays so that each clock is twice delayed, including the once inverted CPE-bar signal with U12A-8 and U12A-6, the two address generator clocks U12A-8 and U12A-6 are separately ORed together with each of the address generator clock gates U9A-6, U9A-8, U10A-6, and U10A-8 (Fig 6B) rather than using the pre-Ored clock signal U21D-8 in order to reduce clock skew.

Clock gating logic will now be discussed in greater detail with reference to Fig 6B. Address decoders U20B and U19B decode the destination address DOC0 to DOC7 to generate a decoded address signal at the outputs of U19B and U20B to select the gated clock channel. This steers the computer load strobe U22B-3 to the addressed register to load that register. The address assignments are set forth in the table entitled DESTINATION SELECT ASSIGNMENTS. The most significant 16-address block is decoded using block decode logic U21D-6 to enable decoders U19B and U20B when the 4-MSBs DOC4, DOC5, DOC6, and DOC7 are all 1-set. The block enable signal U21D-6 enables U19B and U20B with the E1-bar inputs. The most significant address signal DOC3 enables U20B when low, indicated by the E2-bar input, and enables U19B when high, indicated by the E3 input. Consequently, U20B generates the LSH addresses and U19B generates the MSH addresses. One of the 8-addresses for the selected half is selected with the 3-least significant address bits DOC0, DOC1, and DOC2 which go to each address decoder U19B and U20B. The address decoder that has been selected with the DOC3 to DOC7 address signals has one of 8-address output lines low, as determined by the DOC0 to DOC2 least significant address bits. The low output line enables the register clock gating logic to steer the clock to the addressed register. The clock is the negative going strobe U22B-3 generated under computer control. Effectively, the decoded address signals steer the computer strobe to the appropriate register clock input to clock the computer output data word into that register.

Control logic for a display configuration will now be discussed with reference to Fig 6D. Signal U20D-1 is a single pulse clock signal occurring at the leading edge of the line sync signal except when a frame sync pulse or a computer load signal is generated. This clock pulse is generated by U18E-6, as previously described. Disabling of this clock pulse during the frame sync pulse and the computer load period is performed by <sup>F</sup>  
~~ORing~~ together the computer load signal DOA6-bar U20D-12 and the synchronized frame sync signal CFSR1 U20D-11. When either the computer load signal DOA6-bar or the computer frame sync signal CFSR1 are high, NOR gate U20D-13 and inverter U18D-2 apply a high signal to U20D-2, which causes U20D-1 to be low independent of the line sync clock pulse. Only when the computer load signal is in the run state (DOA6-bar is low) and the frame sync signal CSFR1 is low can the line sync clock U20D-1 go high to generate a clock pulse.

The line sync clock U20D-1 is generated from the line sync signal CLS. The CLS signal is latched in flip-flop U22E-12 to generate a resynchronized line sync signal CLSR1. The resynchronized line sync signal CLSR1 is delayed one clock period by latching in flip-flop U22E-15 to generate a delayed resynchronized line sync signal CLSR2. NAND-gate U18E-6 generates a one clock period negative pulse when the resynchronized line sync signal CLSR1 U22E-12 is high and when the delayed resynchronized line sync signal CLSR2 U22E-15 is still low, indicative of the first clock period of the resynchronized line sync signal. Inverter U21E-10 inverts the delayed line sync signal CLSR2 U22E-15 for NANDing with the non-

delayed line sync signal CFSR1 U22E-12 for generation of the one clock period signal U18E-6. Therefore, U20D-1 is a one clock period positive pulse that occurs at the leading edge of each line sync pulse that is disabled by the computer being in the load mode or that is disabled by the frame sync signal. This clock U20D-1 is used to clock the R-registers for updating with the delta parameter at the positive edge and to transfer the updated number in the R-registers to the R-registers at the negative edge.

The computer strobe DOA7 is generated under software control. It is a short positive pulse, typically about 3-microseconds in width. It is NANDed with the computer run signal DOA6 using inverter U18D-6 and NAND-gate U22B-3 to generate a short negative pulse when enabled by the DOA6 run/load-bar signal being low, as indicative of a load command. The negative pulse U22B-3 is used to clock the register that is addressed with the computer destination address signal with decoders U19B and U20B to load data from the computer into that selected register.

Address generator clock gating logic will now be discussed with reference to Fig 6B. This logic is composed of gates U19C, U19D, U18C, U8A, U9A, and U10A. This logic comprises 4-channels of clock logic for the address generation registers, where the clock gating logic for each channel is similar to the clock gating logic for the other 3-channels.

R-register gating logic will now be discussed with reference to Fig 6B.

Gate U19C-1 steers the load strobe U22B-3 to clock the register with the computer generated strobe to load the computer generated parameter into the related register. Steering signal U19C-3 steers the computer pulse U19C-2 to the input of gate U18C-1. Gate U18C-3 combines the two mutually exclusive clock signals, the computer strobe and the line sync strobe to clock the XR-register CXRM with signal U18C-3 for the computer strobe and on the rising edge of the line sync pulse.

Gate U19C-10 steers the load strobe U22B-3 to clock the register with the computer generated strobe to load the computer generated parameter into the related register. Steering signal U19C-9 steers the computer pulse U19C-8 to the input of gate U18C-9. Gate U18C-8 combines the two mutually exclusive clock signals, the computer strobe and the line sync strobe to clock the XR-register CXRL with signal U18C-8 for the computer strobe and on the rising edge of the line sync pulse.

Gate U19D-1 steers the load strobe U22B-3 to clock the register with the computer generated strobe to load the computer generated parameter into the related register. Steering signal U19D-3 steers the computer pulse U19D-2 to the input of gate

6 U18C-4; Gate U18C-6 combines the two mutually exclusive clock signals, the computer strobe and the line sync strobe to clock the YR-register CYRM with signal U18C-6 for the computer strobe and on the rising edge of the line sync pulse.

Gate U19D-13 steers the load strobe U22B-3 to clock the register with the computer generated strobe to load the computer generated parameter into the related register. Steering signal U19D-12 steers the computer pulse U19D-11 to the input of gate U18C-12. Gate U18C-11 combines the two mutually exclusive clock signals, the computer strobe and the line sync strobe to clock the YR-register CYRL with signal U18C-11 for the computer strobe and on the rising edge of the line sync pulse.

P-register clock logic will now be discussed with reference to Fig 6B.

The XP-register clock signal U9A-6 to XP-register CXPM is generated from the inverted XR-register clock signal CXRM U8A-2, the gated write signal U18E-11, the re-addressing clock U12A-8, and the scanout clock U12A-6. The inverted R-register clock signal U8A-2 causes the XP-register CXRM to be clocked with the computer strobe and with the trailing edge of the line sync signal, the inverted clock signal from U18C-3. The gated write signal U18E-11 clocks the XP-registers for each write strobe U22C-11 that loads a parameter into memory in order to advance the address generators to the next address. The gated re-addressing clock signal U12A-6 and the gated scanout signal U12A-8 have been discussed above with reference to Fig 6C.

The XP-register clock signal U9A-8 to XP-register CXPL is generated from the inverted XR-register clock signal CXRL U8A-4, the gated write signal U18E-11, the re-addressing clock U12A-8, and the scanout clock U12A-6. The inverted R-register clock signal U8A-4 causes the XP-register CXRL to be clocked with the computer strobe and with the trailing edge of the line sync signal, the inverted clock signal from U18C-8. The gated write signal U18E-11 clocks the XP-registers for each write strobe U22C-11 that loads a parameter into memory in order to advance the address generators to the next address. The gated re-addressing clock signal U12A-6 and the gated scanout signal U12A-8 have been discussed above with reference to Fig 6C.

The YP-register clock signal U10A-6 to YP-register CYPM is generated from the inverted YR-register clock signal CYRM U8A-6, the gated write signal U18E-11, the re-addressing clock U12A-8, and the scanout clock U12A-6. The inverted YR-register clock signal U8A-6 causes the YP-register CYRM to be clocked with the computer strobe and with the trailing edge of the line sync signal, the inverted clock signal from U18C-6. The gated write signal U18E-11 clocks the YP-registers for each write strobe U22C-11 that loads a parameter into memory in order to advance the address generators to the next address. The gated re-addressing clock signal U12A-6 and the gated scanout signal U12A-8 have been discussed above with reference to Fig 6C.

The YP-register clock signal U10A-8 to YP-register CYPL is generated from the inverted YR-register clock signal CYRL U8A-8, the gated write signal U18E-11, the re-addressing clock U12A-8, and the scanout clock U12A-6. The inverted YR-register clock

signal U8A-8 causes the YP-register CYRL to be clocked with the computer strobe and with the trailing edge of the line sync signal, the inverted clock signal from U18C-11. The gated write signal U18E-11 clocks the YP-registers for each write strobe U22C-11 that loads a parameter into memory in order to advance the address generators to the next address. The gated re-addressing clock signal U12A-6 and the gated scanout signal U12A-8 have been discussed above with reference to Fig 6C.

Gates U22C-11, U18D-12, and U18E-11 provide a write strobe to clock the address registers to advance the address in the address registers in accordance with the delta parameters loaded in the delta registers. This write clock clocking of the address registers is used for writing a sequence of words into memory without the need to reload the address registers, where the address registers are incremented with the write strobe to advance the address from the initially loaded address in accordance with the delta parameters. This write strobe is gated with the DOA5 signal with gate U18E-11 to enable advancing the address generators during the load mode and to disable advancing the address generators during the run mode.

### Address Generators

Two address generator configurations are shown in Figs 6O to 6R. The address generators shown in Figs 6O to 6R provide for overflow detection to gate a clock in accordance with the arrangement shown in Fig 6C. The address generators shown in Figs 6O to 6R do not provide for such overflow detection. In this configuration, overflow detection is enhanced by arranging the adder logic so that the overflow from an adder chip coincides with the desired position of overflow detection. In order to provide this feature, an extra adder chip is used in the address generators of Figs 6O and 6P. The address generators shown in Figs 6Q and 6R do not have such overflow detection and consequently can be implemented with one less adder chip.

The XP-address generator will now be discussed with reference to Fig 6O. Register U8D, U9D, and U5D store the address parameter. Register U17D and U7D store the delta parameter for updating the address parameter. Adders U13D to U16D and U6D add the delta parameter to the address parameter to obtain an updated address parameter. Multiplexers U10D to U12D provide for loading initial conditions into the address register during the load mode and provide for updating the address parameter in the address register in response to the delta parameter in the delta register in the run mode.

The delta parameter initial condition is loaded into the delta register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of delta register U17D and U7D. The CXPS clock provides a clock pulse at the appropriate time, as described with reference to Fig 6B, to

clock the initial conditions into the delta registers.

The address parameter initial condition is loaded into the address register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of address register U8D, U9D, and U5D. The CXPM and CXPL clocks provide clock pulses at the appropriate times, as described with reference to Fig 6B, to clock the initial conditions into the address register.

In the run mode, the address register is clocked with the CXPM and CXPL clocks to update the address parameter in response to the delta parameter. The output of the address register; the Q-outputs of the U8D, U9D, and U5D register; are applied to the A-inputs of adder circuits U13D to U16D and U6D. The output of the delta register; the Q-outputs of the U17D and U7D register; are applied to the B-inputs of adder circuits U13D to U16D and U6D. The output of the adder circuits on the summation lines is the binary sum of the A-inputs from the address register and the B-inputs from the delta register, providing an updated address parameter that is input to the address register through the multiplexers to the D-inputs of the address register. Consequently, each time the address register is clocked, the updated address is loaded into the address register and the updated address that is loaded into the address register is output on the Q-lines from the address register to again be added to the delta parameter with the adders to provide the next updated address to the address register.

The multiplexers U10D, U11D, and U12D multiplex the updated address parameter from the adders into the address register to load the initial conditions into the address register at the beginning of the load mode and to load the updated address parameter from the adders thereafter. The ELS signal from U15A-3, as described with reference to Fig 6D, controls the multiplexer to load initial conditions at the start of a line sync pulse and to enable updating of the address parameter with the delta parameter thereafter.

The adders are connected with the carry output from the preceding stage connected to the carry input of the next subsequent stage for a rapid carry propagating through the adder. The adders are arranged so that the overflow from U15D-9 coincides with the point that divides the scanout bits and the re-addressing bits, where the scanout bits are the three less significant bits and the re-addressing bits are the six more significant bits. The next most significant bit XA3 for the XP address generator is used as the board control bit instead of a re-addressing bit for the XP-address generator. The overflow signal U15D-9 is input to U16A-5 and U17A-5 (Fig 6C) for controlling gating of the clock.

The YP-address generator will now be discussed with reference to Fig 6P. Register U8E, U9E, and U5E store the address parameter. Register U17E and U7E store the delta parameter for updating the address parameter. Adders U13E to U16E and U6E add the delta parameter to the address parameter to obtain an updated address parameter. Multiplexers U10E to U12E provide for loading initial conditions into the address register

during the load mode and provide for updating the address parameter in the address register in response to the delta parameter in the delta register in the run mode.

The delta parameter initial condition is loaded into the delta register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of delta register U17E and U7E. The CYPS clock provides a clock pulse at the appropriate time, as described with reference to Fig 6B, to clock the initial conditions into the delta registers.

The address parameter initial condition is loaded into the address register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of address register U8E, U9E, and U5E. The CYPM and CYPL clocks provide clock pulses at the appropriate times, as described with reference to Fig 6B, to clock the initial conditions into the address register.

In the run mode, the address register is clocked with the CYPM and CYPL clocks to update the address parameter in response to the delta parameter. The output of the address register; the Q-outputs of the U8E, U9E, and U5E register; are applied to the A-inputs of adder circuits U13E to U16E and U6E. The output of the delta register; the Q-outputs of the U17E and U7E register; are applied to the B-inputs of adder circuits U13E to U16E and U6E. The output of the adder circuits on the summation lines is the binary sum of the A-inputs from the address register and the B-inputs from the delta register, providing an updated address parameter that is input to the address register through the

multiplexers to the D-inputs of the address register.

Consequently, each time the address register is clocked, the updated address is loaded into the address register and the updated address that is loaded into the address register is output on the Q-lines from the address register to again be added to the delta parameter with the adders to provide the next updated address to the address register.

The multiplexers U10E, U11E, and U12E multiplex the updated address parameter from the adders into the address register to load the initial conditions into the address register at the beginning of the load mode and to load the updated address parameter from the adders thereafter. The ELS signal from U15A-3, as described with reference to Fig 6D, controls the multiplexer to load initial conditions at the start of a line sync pulse and to enable updating of the address parameter with the delta parameter thereafter.

The adders are connected with the carry output from the preceding stage connected to the carry input of the next subsequent stage for a rapid carry propagating through the adder. The adders are arranged so that the overflow from U15E-9 coincides with the point that divides the scanout bits and the re-addressing bits, where the scanout bits are the three less significant bits and the re-addressing bits are the six more significant bits. The overflow signal U15E-9 is input to U16A-2 and U17A-2 (Fig 6C) for controlling gating of the clock.

The XR-address generator will now be discussed with reference to Fig 6Q. Register U8B, U9B, and U5B store the address parameter. Register U17B and U7B store the delta

parameter for updating the address parameter. Adders U13B to U15B and U6B add the delta parameter to the address parameter to obtain an updated address parameter. Multiplexers U10B to U12B provide for loading initial conditions into the address register during the load mode and provide for updating the address parameter in the address register in response to the delta parameter in the delta register in the run mode.

The delta parameter initial condition is loaded into the delta register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of delta register U17B and U7B. The CXRS clock provides a clock pulse at the appropriate time, as described with reference to Fig 6B, to clock the initial conditions into the delta registers.

The address parameter initial condition is loaded into the address register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of address register U8B, U9B, and U5B. The CXRM and CXRL clocks provide clock pulses at the appropriate times, as described with reference to Fig 6B, to clock the initial conditions into the address register.

In the run mode, the address register is clocked with the CXRM and CXRL clocks to update the address parameter in response to the delta parameter. The output of the address register; the Q-outputs of the U8B, U9B, and U5B register; are applied to the A-inputs of adder circuits U13B to U15B and U6B. The output of the delta register; the Q-outputs of the U17B and U7B register; are applied to the B-inputs of adder circuits U13B to U15B and

U6B. The output of the adder circuits on the summation lines is the binary sum of the A-inputs from the address register and the B-inputs from the delta register, providing an updated address parameter that is input to the address register through the multiplexers to the D-inputs of the address register. Consequently, each time the address register is clocked, the updated address is loaded into the address register and the updated address that is loaded into the address register is output on the Q-lines from the address register to again be added to the delta parameter with the adders to provide the next updated address to the address register.

The multiplexers U10B, U11B, and U12B multiplex the updated address parameter from the adders into the address register to load the initial conditions into the address register during the load mode and to load the updated address parameter from the adders during the run mode. The DOA6-bar signal from U18D-6, as described with reference to Fig 6B, controls the multiplexer to load initial conditions in the load mode and to enable updating of the address parameter with the delta parameter in the run mode.

The adders are connected with the carry output from the preceding stage connected to the carry input of the next subsequent stage for a rapid carry propagating through the adder.

The YR-address generator will now be discussed with reference to Fig 6R. Register U8C, U9C, and U5C store the address parameter. Register U17C and U7C store the delta parameter for updating the address parameter. Adders U13C to U15C and U6C add the delta parameter to the address parameter to

obtain an updated address parameter. Multiplexers U10C to U12C provide for loading initial conditions into the address register during the load mode and provide for updating the address parameter in the address register in response to the delta parameter in the delta register in the run mode.

The delta parameter initial condition is loaded into the delta register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of delta register U17C and U7C. The CYRS clock provides a clock pulse at the appropriate time, as described with reference to Fig 6B, to clock the initial conditions into the delta registers.

The address parameter initial condition is loaded into the address register from the computer output port. The 6-LSBs from the computer output byte are applied to the D-inputs of address register U8C, U9C, and U5C. The CYRM and CYRL clocks provide clock pulses at the appropriate times, as described with reference to Fig 6B, to clock the initial conditions into the address register.

In the run mode, the address register is clocked with the CYRM and CYRL clocks to update the address parameter in response to the delta parameter. The output of the address register; the Q-outputs of the U8C, U9C, and U5C register; are applied to the A-inputs of adder circuits U13C to U15C and U6C. The output of the delta register; the Q-outputs of the U17C and U7C register; are applied to the B-inputs of adder circuits U13C to U15C and U6C. The output of the adder circuits on the summation lines is the binary sum of the A-inputs from the address register and the

B-inputs from the delta register, providing an updated address parameter that is input to the address register through the multiplexers to the D-inputs of the address register. Consequently, each time the address register is clocked, the updated address is loaded into the address register and the updated address that is loaded into the address register is output on the Q-lines from the address register to again be added to the delta parameter with the adders to provide the next updated address to the address register.

The multiplexers U10C, U11C, and U12C multiplex the updated address parameter from the adders into the address register to load the initial conditions into the address register during the load mode and to load the updated address parameter from the adders during the run mode. The DOA6-bar signal from U18D-6, as described with reference to Fig 6B, controls the multiplexer to load initial conditions in the load mode and to enable updating of the address parameter with the delta parameter in the run mode.

The adders are connected with the carry output from the preceding stage connected to the carry input of the next subsequent stage for a rapid carry propagating through the adder.

Output of the address signals will now be discussed with reference to Figs 6O and 6P. The Q-outputs of the XP-address register and YP-address register are routed to the memory for accessing and for loading of information. The address connections between the memory and the address generators are listed in the CABLE CONNECTION TABLE here under the heading CABLE-I BM1,2/BL1 (C1). The Y-address bits Y0 to Y8 and the X-

address bits X0 to X8, including the complemented and uncomplemented X3 memory board address select bit, are listed therein together with source circuits on the control logic board and destination circuits on the memory boards.

In the run mode, the XP-address register and YP-address register are continually clocked with the gated clock, as discussed with reference to Fig 6B; resulting in the address parameters being continually updated with the delta parameters and resulting in the memory being continually addressed by the updated address parameters. The XA3 and XA3-bar memory board select signals are gated with U19A-1 in order to blank the CRT monitor and clear the buffer, as discussed with reference to Fig 6D herein.

## MEMORY BOARDS

A detailed design of one configuration of the memory of the present invention is shown in Figs 6E to 6N. The architecture of one of the memory boards is shown in block diagram form in Fig 6E, is shown in detailed block diagram form in Figs 6F to 6J, and is shown in detailed logical diagram form in Figs 6K to 6N. Commonality of symbols and features in these diagrams provide for cross referencing to different levels of detail between these diagrams. This memory uses Mitsubishi M58725P RAMs having 16,384 (16K) bits per RAM, organized in a 2,048 (2K) words by 8-bits per word configuration. Logical diagrams of RAM chips are shown in Figs 6K to 6N. Each RAM chip comprises 11-address lines A0 to A10, 8-data lines DQ1 to DQ8, a chip select line S-bar, an output enable line OE-bar, and a write pulse line W-bar. The address lines address 1-word out of 2K-words stored in the memory chip. The data lines output the addressed word in the read mode and input a word for storage in the write mode. The data lines are tristate lines that, when in the read mode, output the addressed information when enabled with the chip select line S-bar and, when in the write mode, store the data in the addressed location. The output enable signal G-bar controls inputting and outputting of data from the RAM. When disabled with the chip select line or the output enable line, the data lines are in the floating tristate condition.

The memory control logic is comprised of circuits U19A to U19E, as shown in Fig 6F. Buffers U19A and U19D are non-inverting buffers that buffer the address signals for fanout to the RAMs. In this configuration, 11-address signals YA3B to YA8B and XA4B

to XA8B are fanned out to the address inputs A0 to A10 of the RAMs. Decoders U19B, U19C, and U19E decode the scanout portion of the address word for control of the tristate data lines of the RAMs. Decoder U19B decodes the Y-scanout portion YA0 to YA2 of the scanout portion of the Y-address. Decoders U19C and U19E decode the X-scanout portion XA0 to XA2 of the X-scanout portion of the address. Decoder U19C is enabled for read operations and decoder U19E is enabled for write operations. Each of decoders; U19B, U19C, and U19E; receive 3-input scanout address lines and generate 8-decoded scanout control lines in response to the decoding. Decoded signals MY0-bar to MY7-bar from U19B generate Y-control signals to select one column of 8-RAMs with the chip select pin S-bar. Decoded signals MX0R-bar to MX7R-bar from U19C generate X-control signals in the read mode to select one row of 8-RAMs with the output enable pin G-bar. Decoded signals MX0W-bar to MX7W-bar from U19E generate X-control signals in the write mode to select one row of 8-RAMs with the output enable pin G-bar. Therefore, selection of a column of 8-RAMs with the Y-scanout signals and selection of a row of 8-RAMs with the X-scanout signals selects a single RAM common to both the selected row and the selected column for read and write operations.

The decoders have gating signals E1-bar, E2-bar, and E3. Signals E1-bar and E2-bar are permanently enabled on decoders U19B and U19C with a ground connection and signal E3 is permanently enabled on decoder U19E with a pullup connection. Signal E3 on decoder U19B is used to place the RAMs on the non-selected memory board in the standby mode for reduced power

consumption. This is achieved by disabling U19B on the non-selected board and enabling U19B on the selected board with the XA3 and XA3-bar signals, which select the board to be utilized. Disabling U19B on the non-selected board disables the Y-scanout signals to the chip select pin S-bar, which in turn places the non-selected RAMs into the standby mode for reduced power consumption. The XA3 and XA3-bar gating of the U19B decoder is not necessary for memory operation, but is used for reduction of power consumption. Pin E3 on decoder U19C and pin E2-bar on decoder U19E are controlled with the DIEN-bar signal which is derived from the computer run/load mode signal for enabling decoder U19C in the run mode to read from memory, to disable decoder U19E in the run mode to prevent writing into memory, to enable decoder U19E in the load mode to write into memory and to disable decoder U19C in the load mode to read from memory. When decoder U19E is enabled in the load mode, the write pulse W-bar controls decoder U19E; where decoder U19E effectively steers the W-bar write pulse to the one of eight decoded signal lines MX0W-bar to MX7W-bar under control of the XA0, XA1, and XA2 scanout address signals.

A discussion will now be provided relative to Figs 6G to 6N to illustrate the logical design of the memory. Figs 6G to 6K each show 16-RAMs organized in 2-logical columns and constructed on one row on a memory board, where each memory board has 4-groups of 16-RAMs each shown in one of Figs 6G to 6J. The manner in which these 4-pairs of logical columns connect together is shown in Fig 6E and is shown by the interconnections between Figs 6G to 6J and Figs 6K to 6N and by the discussions hereinafter.

All RAMs are addressed by the same 11-address lines; shown as the address bus to the A-input of each RAM in Figs 6G to 6J and shown in greater detail as the address bus to the A0 to A10 inputs of RAMs U8 and U16 for each pair of logical columns in Figs 6K to 6N. The addresses are placed on the address bus by U19A and U19D (Fig 6F) and are routed to all 16-RAMs on each of the RAM groups (Figs 6G to 6J) to excite the RAM address inputs A (Figs 6G to 6J) and A0 to A10 (Figs 6K to 6N).

Each logical column of RAMs is selected by a single one of the eight Y-scanout signals from U19B (Fig 6F), shown connected to the S-bar pin of each RAM in the logical column in Figs 6G to 6J and shown in greater detail to the S-bar inputs of RAMs U8 and U16 for each pair of logical columns in Figs 6K to 6N.

Each logical row of RAMs is selected by a single one of the eight X-scanout signals; MX0R to MX7R in the read mode to the output enable pin G-bar and MX0W to MX7W in the write mode to the W-bar from U19C and U19E respectively (Fig 6F). Each of these signals are shown connected to the pair of RAM in the logical row for each pair of logical columns in Figs 6G to 6J and shown in greater detail for RAMs U8 and U16 for each pair of logical columns in Figs 6K to 6N. Each of these X-scanout signals enable the same pair of RAMs in each of the 4-pairs of logical columns (Figs 6G to 6J and Figs 6K to 6N).

All RAMs in the pair of columns shown in each of Figs 6G to 6J <sup>and</sup> Figs 6K to 6N have the corresponding data bus pins collected together; shown as the data bus to the D-input of each RAM in Figs 6G to 6J and shown in greater detail as the data bus

to the D0 to D7 pins of RAMs U8 and U16 for each pair of logical columns in Figs 6K to 6N. The data bus and control signals are connected to a pair of Intel 8216 bus interface chips for each pair of logical columns, as shown in simplified form in Figs 6G to 6J and in detailed form in Figs 6K to 6N. The operation of the data bus interface is discussed in greater detail hereinafter.

In view of the above, all RAMs are addressed with the same address signals and one RAM that is at the intersection of the enabled X-row scanout signal and the enabled Y-column scanout signal alone is permitted to place the addressed word on the data bus in the read mode and alone is written into at the addressed location from the data bus.

In the read mode, one RAM is enabled to output the addressed word onto the system data bus. This can be implemented by busing together the corresponding 8-data lines from each RAM. However, busing together a large number of RAM data lines, such as 128-RAM data lines in this configuration, can result in reduced RAM speed, such as due to bus and chip capacitance. Therefore, Intel 8216 bi-directional bus drivers are provided to isolate groups of RAMs from the system data bus and from other groups of RAMs.

The system data bus interface will now be described with reference to Figs 6K to 6N. Fig 6K shows the data bus interface for row-A on each of the two memory boards. Fig 6L shows the data bus interface for row-B on each of the two memory boards. Fig 6M shows the data bus interface for row-C on each of the two memory boards. Fig 6N shows the data bus interface for row-D on each of the two memory boards.

For this configuration, 16-RAMs are bused together through each system data bus interface circuit, comprising two Intel 8216 components. The bi-directional signals DB0, DB1, DB2, and DB3 are connected to the RAM data bus and the uni-directional signals DI0 and DO0, DI1 and DO1, DI2 and DO2, and DI3 and DO3 are connected to the uni-directional system bus for reading from RAM through the DO0 to DO3 unidirectional outputs and for writing into RAM on the DI0 to DI3 unidirectional inputs. A pair of Intel 8216 4-bit bus drivers are used to control the 8-data bus lines for the RAM data bus.

The Intel 8216s are controlled with the board select signal XA3 or XA3-bar and the scanout address select signals for the 2-columns of RAMs associated with the Intel 8216 circuits. The board select signal, XA3-bar for memory board-1 and XA3 for memory board-2, control all of the Intel 8216s on the board. Therefore, the Intel 8216s on the enabled board are partially enabled to connect the RAM data buses on that board to the system bus and the Intel 8216 on the disabled board are fully disabled to disconnect the RAM datbuses on that board from the system bus. Similarly, the column select signals for the 2-columns of RAMs connected to the particular Intel 8216s are controlled with the column select signals so that the column select signal enabling a column of RAMs will also partially enable the Intel 8216s associated with that column to connect the selected column data bus to the system data bus. Because 2-columns of RAMs are connected to each Intel 8216 circuit, the related column select signals are ORed together with a NAND gate;

where the column select signals are in complement logic form and consequently a NAND gate can perform an OR function; and are then ANDed with the board select signal XA3 or XA3-bar in a second NAND gate in non-complement logic form to generate the control signal in complement logic form as needed for the Intel 8216 chip select. The Intel 8216 circuits are steered with the DIEN-bar control, which is connected to the DIEN signal generated with U21E-6 (Fig 6B). Therefore, in the run mode; the DIEN signal controls the Intel 8216s to connect the RAM data bus through the D00 to D03 buffers to output the RAM signals onto the system output data bus and, in the load mode; the DIEN signal controls the Intel 8216s to connect the RAM data bus through the D10 to D13 buffers to input the write signals from the system data bus to the RAMs.

Row A on the board has 2-logical columns of RAMs (Figs 6G and 6K), comprising the first column with RAMs U1A to U8A and the second column with RAMs U9A to U16A. As shown in greater detail with reference to Figs 6K to 6N; the first column is selected with the MY0-bar column select scanout signal and the second column is selected with the MY1-bar column select signal. The 2-column select signals are ORed together with U18E-3 and are ANDed with the board select signal with U18E-6 to enable U17A and U18A.

Row B on the board has 2-logical columns of RAMs (Figs 6H and 6L), comprising the third column with RAMs U1B to U8B and the fourth column with RAMs U9B to U16B. As shown in greater detail with reference to Figs 6K to 6N; the third column is selected with the MY2-bar column select scanout signal and the fourth column is selected with the MY3-bar column select signal. The 2-

column select signals are ORed together with U18E-8 and are ANDed with the board select signal with U18E-11 to enable U17B and U18B.

Row C on the board has 2-logical columns of RAMs (Figs 6I and 6M), comprising the fifth column with RAMs U1C to U8C and the sixth column with RAMs U9C to U16C. As shown in greater detail with reference to Figs 6K to 6N; the fifth column is selected with the MY4-bar column select scanout signal and the sixth column is selected with the MY5-bar column select signal. The 2-column select signals are ORed together with U17E-3 and are ANDed with the board select signal with U17E-6 to enable U17C and U18C.

Row D on the board has 2-logical columns of RAMs (Figs 6J and 6N), comprising the seventh column with RAMs U1D<sup>to</sup><sub>6</sub> U8D and the eighth column with RAMs U9D to U16D. As shown in greater detail with reference to Figs 6K to 6N; the seventh column is selected with the MY6-bar column select scanout signal and the eighth column is selected with the MY7-bar column select signal. The 2-column select signals are ORed together with U17E-8 and are ANDed with the board select signal with U17E-11 to enable U17D and U18D.

## BUFFER BOARD

### General

The buffer board is implemented with a buffer to buffer the outputs of memory and to apply these outputs to the rear-end board (Fig 6A). The buffer can be implemented with various forms of buffer memories, such as double buffers and FIFOs.

Alternately, the buffer can be eliminated and the memory output can be routed directly to the rear-end board.

The CABLE CONNECTION TABLE provided herein illustrates an arrangement for connecting the buffer board <sup>"</sup> between the memory and logic boards and the rear-end board. The buffer board receives memory information and a gated clock from the memory and logic boards to clock the memory information into the buffer and receives a rear-end board clock to clock the buffered information into the rear-end board, as generally discussed with reference to Fig 6A.

Alternately, the memory and logic boards can provide the unbuffered information directly to the rear-end board without an intervening buffer board for clocking the memory information into the DACs on the rear-end board with the gated clock from the logic board.

#### 4/3 Buffer Implementation

The 4/3 buffer will now be discussed with reference to Figs 6W to 6AD. A precessional approach is shown controlled with counter U8A decoded with decoder U12B to identify which of the 4-precessional conditions exists for each line-buffer (Fig 6W). This decoded number controls which of the 4-lines of buffer memory are selected for each of the 4-operational positions; the input line buffer, the output line-buffer channel-1, the output line-buffer channel-2, and the output line-buffer channel-3 positions. As the precessional counter sequences through its 4-states, the precessional decoder sequences through the 4-select signals to control sequential precessional operation. The selection logic is shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE, where the input precessional counter states are shown decoded into the decoded states and shown further decoded into the single input line-buffer select signal, the 3-output line-buffer select signal, and the output weight select signal.

BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE

CNTR STATE	<u>INPUT</u>		OUT ADD. COUNTER	<u>OUTPUT</u>				WEIGHT
	INPUT DATA	IN ADD. COUNTER		CH-I	CH-II	CH-III		
	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD	ABCD
00	1000	1000	0111	0100	0010	0001	0010	
01	0100	0100	1011	0010	0001	1000	0001	
10	0010	0010	1101	0001	1000	0100	1000	
11	0001	0001	1110	1000	0100	0010	0100	

NOTES:

- 1) INPUT DATA assignments are the same as ADD. COUNTER assignments.
- 2) OUTPUT CH-II assignments are the same as OUTPUT WEIGHT assignments.
- 3) INPUT ADD. COUNTER assignments are the complement of OUTPUT ADD. COUNTER assignments.

The 4/3rds buffer is shown implemented with two address counters, an input address counter controlled by the image processor and the image memory clock and an output address counter controlled by the CRT monitor clock. The input address counter controls a single line-buffer that is being loaded by the image processor. The output address counter controls three line-buffers outputting to the spatial filter. Address multiplexing logic consists of multiplexers that select either the input counter or the output counter for the address input of each line-buffer RAM in accordance with the input columns in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE. Similarly, this logic selects the write mode for the line buffer being loaded and selects the read mode for the 3-line-buffers being unloaded.

Similar to the address selection discussed above, the input and output signals to the RAM line-buffer being loaded and from the RAM line-buffer being unloaded are shown implemented with input multiplexers U4F, U5F, U6F, U8F, U9F, U10F, U12F, U13F, U14F, U16F, U17F, and U18F and output multiplexers U3D, U4D, U5D, U6D, U7D, U8D, U9D, U10D, U11D, U12D, U13D, U14D, U15D, U16D, U17D, and U18D. The single line-buffer, selected for inputting and operating under control of the input address counter, receives input data from the image processor for storing therein. The 3-line buffers, selected for outputting and operating under control of the output address counter, generates output data to the post-processor for exciting the display. The output channel of each of the line buffers is controlled for outputting to each of the 3-output channels under control of the precessional decoder signals. Each of the 3-line buffers selected for

outputting are selected for outputting to different output channels; channel-1, channel-2, and channel-3; under control of the precessional decoder signals; as shown in the OUTPUT columns of the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE. Similarly, the spatial filtering weight is selected from the channel-2 line-buffer, as shown in the WEIGHT column of the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE.

In view of the above and in view of the referenced schematics and block diagrams, a buffer memory is provided that inputs pixel information to any one of 4-line-buffers and outputs pixel information from any one of 4-line-buffers to any one of 3-output channels. Also, sequential operation is provided that sequentially applies each of the 4-line buffers to each of the positions of input buffer, channel-1 output buffer, channel-2 output buffer, and channel-3 output buffer for loading of a single line into a line-buffer and simultaneously for outputting of 3-parallel lines into a post-processor.

The buffer memory is shown implemented to store both weight address information for weight determination and pixel intensity information. The pixel intensity information is loaded into an interface pipeline (kernel) register, such as for use with the post processor, such as a spatial processor. The weight address information is converted to weight information with a table lookup in a weight-RAM. Weight address information is stored in the line-buffer to implement interpolation in the post-processor; which, in one configuration may be more efficiently implemented with weight address information. Alternately, the weight address

information can be converted to weight information before loading into the line buffers, such as with a weight RAM, for storing of weight information instead of weight address information in the buffer memory.

A register interface is provided <sup>11</sup> between the buffer memory and the post-processor. The registers may be considered to be implemented in a pipeline form to enhance performance and to provide stable latched signals for the post-processor. The registers are implemented to store 9-pixels in a 9-pixel kernel arrangement for processing with a post-processor. Also, a weight RAM is provided that stores weight information in response to weight address information from the buffer memory.

The memory components are Intel 2149H RAMs. The logic components are 7400 series logic, particularly 74LS, 74ALS, and 74AS circuits; such as produced by Texas Instruments. Alternately, the buffer memory can be implemented with other components.

### Buffer Memory Address Counters

The buffer memory is composed of one line buffer selected as an input line buffer and 3-line buffers selected as output line buffers. Separate address counters are used to generate addresses for the input line buffer and to generate addresses for the output line buffers so that loading of data into the input line buffer can proceed asynchronously with respect to outputting of data from the output line buffers. The output address counter is discussed relative to Fig 6X and the input address counter is discussed relative to Fig 6W below.

Control logic for the address counters is shown in Fig 6X, comprising flip-flops U1A and gates U1B-1, U2B-3, AND U2B-6. Gate U1B-1 receives and inverts the line sync signal CLS for loading into flip-flops U1A. The inverted line sync pulse is shifted through flip-flops U1A-2 and U1A-7 and the leading edge is detected with gate U2B-3 and the trailing edge is detected with gate U2B-6. Leading edge detection is performed by NANDing the once delayed complemented and the twice delayed uncomplemented shifted CLS-signal with gate U2B-3. Trailing edge detection is performed by NANDing the once delayed uncomplemented and the twice delayed complemented shifted CLS-signal with gate U2B-6. Therefore, gate U2B-3 will generate a 1-clock period negative going pulse at the leading edge of the CLS-signal and gate U2B-6 will generate a 1-clock period negative going pulse at the trailing edge of the CLS-signal for synchronizing the input address counter and the output address counter to the line sync signal, respectively.

As discussed above, the input line buffer can be controlled to commence loading of information at the beginning of the line sync period to provide extra time to load the line buffer and the output line buffer can be controlled to commence outputting of information at the end of the line sync period to synchronize outputting of pixel information with completion of the blanking pulse.

Clock fanout logic is shown in Fig 6X, where AND gate U7B-3 receives the complemented early clock CPE-bar and provides a non-inverting delay and inverters U8B-2, U8B-4, and U8B-6 provide the inverted and delayed CPD clock signal.

The output address counter will now be discussed with reference to Fig 6X. The output address counter is composed of counters U2A, U3A, and U4A. This counter is connected as a synchronous 12-bit counter by connecting the TC-signal on pin-15, the CEP-signal on pin-7, and the CET-signal on pin-10 between the different stages in the manner shown. The output counter is clocked under control of the output pixel clock CPE-bar inverted and delayed to the CPD clock signal to generate 10-address signals to the output address bus (or read address bus) to the buffer memories. The output address counter is cleared with the trailing edge pulse from U2B-6 to the counter clear input PE-bar on pin-9 at the end of the line blanking period to initiate address counting from the first address in the buffer memory coincident with the beginning of the line blanking CLS-signal.

The input address counter will now be discussed with reference to Fig 6W. The input address counter is composed of counters U5A, U6A, and U7A. This counter is connected as a synchronous 12-bit counter by connecting the TC-signal on pin-15, the CEP-signal on pin-7, and the CET-signal on pin-10 between the different stages in the manner shown. The input counter is clocked under control of the input pixel gated clock CPG C3-22 to generate 10-address signals to the input address bus (or write address bus) to the buffer memories. The CPG clock is the gated clock generated on the control logic board with gate U21D-8. The input address counter is cleared with the leading edge pulse from U2B-3 to the counter clear input PE-bar on pin-9 at the beginning of the line blanking period to initiate address counting from the first address in the buffer memory coincident with the end of the line blanking CLS-signal.

The buffer memory multiplexer control will now be discussed with reference to Fig 6W. The multiplexer logic has been discussed with reference to the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE above. Counter U8A is a modulo-3 counter, counting from 0 to 3 and then overflowing to 0 again. It counts in response to the leading edge pulse from U2B-3, inverted with U13B-10, and applied to the CEP and CET inputs to increment counter U8A. Consequently, counter U8A is incremented at the beginning of each line sync pulse for changing the buffer memory multiplexing on a scanline-by-scanline basis. The modulo-3 count is decoded by decoder U12B to generate one of four mode signals; F0-bar, F1-bar, F2-bar, and F3-bar in response to the state of counter U8A. The decoded mode signal controls the buffer memory

multiplexer in accordance with the assignments in the BUFFER  
MEMORY MULTIPLEXER ASSIGNMENT TABLE.

### Line Buffer Memory

The line buffer memory is composed of 4-line buffers shared to implement a multiple buffer configuration. These 4-line buffers are similar in design and operate under control of multiplexer signals and address signals.

The first line buffer will now be discussed with reference to Figs 6Y and 6Z. Four RAM circuits U3E to U6E are addressed through tristate multiplexers U3C to U6C. Multiplexers U3C and U4C select the write address from the write bus (or input bus) and multiplexers U5C and U6C select the read address from the read bus (or output bus) under control of the mode control signal F0-bar U12B-4 in accordance with the logic shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE. Multiplexers U3C and U4C are selected with the F0-bar signal and multiplexers U5C and U6C are selected with the F0-bar signal complemented with U13B-2. The selected address is applied to the address inputs on RAMs U3E to U6E. The write multiplexer U4C also multiplexes the write pulse to U4C-11 and to the write inputs to the RAMs WE-bar on pin-10. When the write multiplexer U3C and U4C is selected, the write address is applied to RAMs U3E to U6E and the write pulse CPG is applied to the write input WE-bar of the RAMs. When the read multiplexer U5C and U6C is selected, the read address is applied to RAMs U3E to U6E and the write pin-10 WE-bar is pulled up to select read operations.

The RAM input and output lines I00 to I03 are shared between outputting of information in the read mode and inputting of information in the write mode. Input information is selected with line buffers U4F to U6F, which apply input information obtained

from the input bus to the RAM IO lines in the write mode and which are tristate-disabled to disconnect the input information on the input bus from the RAM IO lines in the read mode. Line buffers U4F to U6F are controlled with the F0-bar multiplexer control signal to connect the input bus to the RAM IO lines when the F0-bar signal enables the write address and the write pulse through multiplexers U3C and U4C and to disconnect the input bus from the RAM IO lines when the F0-bar signal enables the read address and disables the write address and write pulse. Hence, line buffers U4F to U6F are enabled when the RAM channel is in the write mode and are disabled when the RAM channel is in the read mode. As can be seen from the schematic, line buffers U4F to U6F apply the 8-intensity bits from image memory over the buffer input bus to the inputs of RAMs U3E and U4E and apply the weight addresses from the address generators over the buffer input bus to the inputs of RAMs U5E and U6E, with 2-spare bits presently unassigned.

Output information is multiplexed onto the buffer output buses; identified as the channel-I bus, the channel-II bus, the channel-III bus, and the weight bus. The 3-channel buses comprising the pixel intensities from the 3-output line buffers are applied to the 3-channels of kernel registers. The weight bus, comprising the weight addresses of the selected channel, is applied to the single channel of weight registers U22A and U22B for accessing the weight RAM.

The 8-intensity lines from RAMs U3E and U4E are applied to all 3-channels through different multiplexers so that the intensity lines can be selected for one of the 3-output channels or for none of the 3-output channels under control of multiplexers U3D to U6D. Because each intensity parameter and each output channel has 8-bits, because each multiplexer U3D to U6D has 6-bits, and because the 74LS367 multiplexers can each control <sup>A</sup>~~separately~~ <sup>A</sup> 4-bits and 2-bits, the 8-bits selected for one of 3-channels are partitioned between a pair of multiplexers. For example, the intensity bits from RAMs U3E and U4E are controlled with multiplexers U3D and U4D for channel-I, with multiplexers U4D and U5D for channel-II, and with multiplexers U5D and U6D for channel-III. The specific partitioning of intensity bits and multiplexers is shown in the schematic diagram. The multiplexing of the intensity bits onto the channel-I bus is controlled with the F1-bar multiplexer control signal to control all of multiplexer U3D and 2-bits of multiplexer U4D. The multiplexing of the intensity byte onto the channel-II bus is controlled with the F2-bar multiplexer control signal to control 4-bits of multiplexer U4D and 4-bits of multiplexer U5D. The multiplexing of the intensity byte onto the channel-III bus is controlled with the F3-bar multiplexer control signal to control 2-bits of multiplexer U5D and all 6-bits of multiplexer U6D. Consequently, multiplexers U3D to U6D multiplex the 8-intensity bits from RAMs U3E and U4E onto one of the 3-output channel buses under control of the multiplexer control signals in accordance with the definitions shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE.

The 6-weight address lines are applied to the weight bus under control of multiplexer U6B as output from RAMs U5E and U6E. This first RAM buffer line outputs to the weight bus under control of the F2-bar signal. Consequently, when the first line buffer is selected with the F2-bar signal to output the stored intensity onto the channel-II bus through multiplexers U4D and U5D, the weight addresses from RAMs U5E and U6E are applied to the weight bus through multiplexer U6B under control of the F2-bar signal.

The second line buffer will now be discussed with reference to Figs 6AA and 6AB. Four RAM circuits U7E to U10E are addressed through tristate multiplexers U7C to U10C. Multiplexers U7C and U8C select the write address from the write bus (or input bus) and multiplexers U9C and U10C select the read address from the read bus (or output bus) under control of the mode control signal F1-bar U12B-5 in accordance with the logic shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE. Multiplexers U7C and U8C are selected with the F1-bar signal and multiplexers U9C and U10C are selected with the F1-bar signal complemented with U13B-4. The selected address is applied to the address inputs on RAMs U7E to U10E. The write multiplexer U8C also multiplexes the write pulse to U8C-11 and to the write inputs to the RAMs WE-bar on pin-10. When the write multiplexer U7C and U8C is selected, the write address is applied to RAMs U7E to U10E and the write pulse CPG is applied to the write input WE-bar of the RAMs. When the read multiplexer U9C and U10C is selected, the read address is applied to RAMs U7E to U10E and the write pin-10 WE-bar is pulled up to

select read operations.

The RAM input and output lines I00 to I03 are shared between outputting of information in the read mode and inputting of information in the write mode. Input information is selected with line buffers U8F to U10F, which apply input information obtained from the input bus to the RAM IO lines in the write mode and which are tristate-disabled to disconnect the input information on the input bus from the RAM IO lines in the read mode. Line buffers U8F to U10F are controlled with the F1-bar multiplexer control signal to connect the input bus to the RAM IO lines when the F1-bar signal enables the write address and the write pulse through multiplexers U7C and U8C and to disconnect the input bus from the RAM IO lines when the F1-bar signal enables the read address and disables the write address and write pulse. Hence, line buffers U8F to U10F are enabled when the RAM channel is in the write mode and are disabled when the RAM channel is in the read mode. As can be seen from the schematic, line buffers U8F to U10F apply the 8-intensity bits from image memory over the buffer input bus to the inputs of RAMs U7E and U8E and apply the weight addresses from the address generators over the buffer input bus to the inputs of RAMs U9E and U10E, with 2-spare bits presently unassigned.

Output information is multiplexed onto the buffer output buses; identified as the channel-I bus, the channel-II bus, the channel-III bus, and the weight bus. The 3-channel buses comprising the pixel intensities from the 3-output line buffers are applied to the 3-channels of kernel registers. The weight bus, comprising the weight addresses of the selected channel, is

applied to the single channel of weight registers U22A and U22B for accessing the weight RAM.

The 8-intensity lines from RAMs U7E and U8E are applied to all 3-channels through different multiplexers so that the intensity lines can be selected for one of the 3-output channels or for none of the 3-output channels under control of multiplexers U7D to U10D. Because each intensity parameter and each output channel has 8-bits, because each multiplexer U7D to U10D has 6-bits, and because the 74LS367 multiplexers can each control separately 4-bits and 2-bits, the 8-bits selected for one of 3-channels are partitioned between a pair of multiplexers. For example, the intensity bits from RAMs U7E and U8E are controlled with multiplexers U7D and U8D for channel-I, with multiplexers U8D and U9D for channel-II, and with multiplexers U9D and U10D for channel-III. The specific partitioning of intensity bits and multiplexers is shown in the schematic diagram. The multiplexing of the intensity bits onto the channel-I bus is controlled with the F2-bar multiplexer control signal to control all of multiplexer U7D and 2-bits of multiplexer U8D. The multiplexing of the intensity byte onto the channel-II bus is controlled with the F3-bar multiplexer control signal to control 4-bits of multiplexer U8D and 4-bits of multiplexer U9D. The multiplexing of the intensity byte onto the channel-III bus is controlled with the F0-bar multiplexer control signal to control 2-bits of multiplexer U9D and all 6-bits of multiplexer U10D. Consequently, multiplexers U7D to U10D multiplex the 8-intensity bits from RAMs U7E and U8E onto one of the 3-output channel buses under control

of the multiplexer control signals in accordance with the definitions shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE.

The 6-weight address lines are applied to the weight bus under control of multiplexer U10B as output from RAMs U9E and U10E. This first RAM buffer line outputs to the weight bus under control of the F3-bar signal. Consequently, when the first line buffer is selected with the F3-bar signal to output the stored intensity onto the channel-II bus through multiplexers U8D and U9D, the weight addresses from RAMs U9E and U10E are applied to the weight bus through multiplexer U10B under control of the F3-bar signal.

The third line buffer will now be discussed with reference to Figs 6AC and 6AD. Four RAM circuits U11E to U14E are addressed through tristate multiplexers U11C to U14C. Multiplexers U11C and U12C select the write address from the write bus (or input bus) and multiplexers U13C and U14C select the read address from the read bus (or output bus) under control of the mode control signal F2-bar U12B-6 in accordance with the logic shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE. Multiplexers U11C and U12C are selected with the F2-bar signal and multiplexers U13C and U14C are selected with the F2-bar signal complemented with U13B-6. The selected address is applied to the address inputs on RAMs U11E to U14E. The write multiplexer U12C also multiplexes the write pulse to U12C-11 and to the write inputs to the RAMs WE-bar on pin-10. When the write multiplexer U11C and U12C is selected, the write address is applied to RAMs U11E to U14E and the write pulse CPG is applied to the write input WE-bar of the RAMs. When

the read multiplexer U13C and U14C is selected, the read address is applied to RAMs U11E to U14E and the write pin-10 WE-bar is pulled up to select read operations.

The RAM input and output lines I00 to I03 are shared between outputting of information in the read mode and inputting of information in the write mode. Input information is selected with line buffers U12F to U14F, which apply input information obtained from the input bus to the RAM IO lines in the write mode and which are tristate-disabled to disconnect the input information on the input bus from the RAM IO lines in the read mode. Line buffers U12F to U14F are controlled with the F2-bar multiplexer control signal to connect the input bus to the RAM IO lines when the F2-bar signal enables the write address and the write pulse through multiplexers U11C and U12C and to disconnect the input bus from the RAM IO lines when the F2-bar signal enables the read address and disables the write address and write pulse. Hence, line buffers U12F to U14F are enabled when the RAM channel is in the write mode and are disabled when the RAM channel is in the read mode. As can be seen from the schematic, line buffers U12F to U14F apply the 8-intensity bits from image memory over the buffer input bus to the inputs of RAMs U11E and U12E and apply the weight addresses from the address generators over the buffer input bus to the inputs of RAMs U13E and U14E, with 2-spare bits presently unassigned.

Output information is multiplexed onto the buffer output buses; identified as the channel-I bus, the channel-II bus, the channel-III bus, and the weight bus. The 3-channel buses

comprising the pixel intensities from the 3-output line buffers are applied to the 3-channels of kernel registers. The weight bus, comprising the weight addresses of the selected channel, is applied to the single channel of weight registers U22A and U22B for accessing the weight RAM.

The 8-intensity lines from RAMs U11E and U12E are applied to all 3-channels through different multiplexers so that the intensity lines can be selected for one of the 3-output channels or for none of the 3-output channels under control of multiplexers U11D to U14D. Because each intensity parameter and each output channel has 8-bits, because each multiplexer U11D to U14D has 6-bits, and because the 74LS367 multiplexers can each control separately 4-bits and 2-bits, the 8-bits selected for one of 3-channels are partitioned between a pair of multiplexers. For example, the intensity bits from RAMs U11E and U12E are controlled with multiplexers U11D and U12D for channel-I, with multiplexers U12D and U13D for channel-II, and with multiplexers U13D and U14D for channel-III. The specific partitioning of intensity bits and multiplexers is shown in the schematic diagram. The multiplexing of the intensity bits onto the channel-I bus is controlled with the F3-bar multiplexer control signal to control all of multiplexer U11D and 2-bits of multiplexer U12D. The multiplexing of the intensity byte onto the channel-II bus is controlled with the F0-bar multiplexer control signal to control 4-bits of multiplexer U12D and 4-bits of multiplexer U13D. The multiplexing of the intensity byte onto the channel-III bus is controlled with the F1-bar multiplexer control signal to control 2-bits of multiplexer U13D and all 6-bits of multiplexer U14D.

Consequently, multiplexers U11D to U14D multiplex the 8-intensity bits from RAMs U11E and U12E onto one of the 3-output channel buses under control of the multiplexer control signals in accordance with the definitions shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE.

The 6-weight address lines are applied to the weight bus under control of multiplexer U14B as output from RAMs U13E and U14E. This first RAM buffer line outputs to the weight bus under control of the F0-bar signal. Consequently, when the first line buffer is selected with the F0-bar signal to output the stored intensity onto the channel-II bus through multiplexers U12D and U13D, the weight addresses from RAMs U13E and U14E are applied to the weight bus through multiplexer U14B under control of the F0-bar signal.

The fourth line buffer will now be discussed with reference to Figs 6AE and 6AF. Four RAM circuits U15E to U18E are addressed through tristate multiplexers U15C to U18C. Multiplexers U15C and U16C select the write address from the write bus (or input bus) and multiplexers U17C and U18C select the read address from the read bus (or output bus) under control of the mode control signal F3-bar U12B-7 in accordance with the logic shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE. Multiplexers U15C and U16C are selected with the F3-bar signal and multiplexers U17C and U18C are selected with the F3-bar signal complemented with U13B-8. The selected address is applied to the address inputs on RAMs U15E to U18E. The write multiplexer U16C also multiplexes the write pulse to U16C-11 and to the write inputs to the RAMs WE-bar

on pin-10. When the write multiplexer U15C and U16C is selected, the write address is applied to RAMs U15E to U18E and the write pulse CPG is applied to the write input WE-bar of the RAMs. When the read multiplexer U17C and U18C is selected, the read address is applied to RAMs U15E to U18E and the write pin-10 WE-bar is pulled up to select read operations.

The RAM input and output lines IO0 to IO3 are shared between outputting of information in the read mode and inputting of information in the write mode. Input information is selected with line buffers U16F to U18F, which apply input information obtained from the input bus to the RAM IO lines in the write mode and which are tristate-disabled to disconnect the input information on the input bus from the RAM IO lines in the read mode. Line buffers U16F to U18F are controlled with the F3-bar multiplexer control signal to connect the input bus to the RAM IO lines when the F3-bar signal enables the write address and the write pulse through multiplexers U15C and U16C and to disconnect the input bus from the RAM IO lines when the F3-bar signal enables the read address and disables the write address and write pulse. Hence, line buffers U16F to U18F are enabled when the RAM channel is in the write mode and are disabled when the RAM channel is in the read mode. As can be seen from the schematic, line buffers U16F to U18F apply the 8-intensity bits from image memory over the buffer input bus to the inputs of RAMs U15E and U16E and apply the weight addresses from the address generators over the buffer input bus to the inputs of RAMs U17E and U18E, with 2-spare bits presently unassigned.

Output information is multiplexed onto the buffer output buses; identified as the channel-I bus, the channel-II bus, the channel-III bus, and the weight bus. The 3-channel buses comprising the pixel intensities from the 3-output line buffers are applied to the 3-channels of kernel registers. The weight bus, comprising the weight addresses of the selected channel, is applied to the single channel of weight registers U22A and U22B for accessing the weight RAM.

The 8-intensity lines from RAMs U15E and U16E are applied to all 3-channels through different multiplexers so that the intensity lines can be selected for one of the 3-output channels or for none of the 3-output channels under control of multiplexers U15D to U18D. Because each intensity parameter and each output channel has 8-bits, because each multiplexer U15D to U18D has 6-bits, and because the 74LS367 multiplexers can each control separately 4-bits and 2-bits, the 8-bits selected for one of 3-channels are partitioned between a pair of multiplexers. For example, the intensity bits from RAMs U15E and U16E are controlled with multiplexers U15D and U16D for channel-I, with multiplexers U16D and U17D for channel-II, and with multiplexers U17D and U18D for channel-III. The specific partitioning of intensity bits and multiplexers is shown in the schematic diagram. The multiplexing of the intensity bits onto the channel-I bus is controlled with the F0-bar multiplexer control signal to control all of multiplexer U15D and 2-bits of multiplexer U16D. The multiplexing of the intensity byte onto the channel-II bus is controlled with the F1-bar multiplexer control signal to control 4-bits of multiplexer U16D and 4-bits of multiplexer U17D. The

multiplexing of the intensity byte onto the channel-III bus is controlled with the F2-bar multiplexer control signal to control 2-bits of multiplexer U16D and all 6-bits of multiplexer U17D. Consequently, multiplexers U15D to U18D multiplex the 8-intensity bits from RAMs U15E and U16E onto one of the 3-output channel buses under control of the multiplexer control signals in accordance with the definitions shown in the BUFFER MEMORY MULTIPLEXER ASSIGNMENT TABLE.

The 6-weight address lines are applied to the weight bus under control of multiplexer U18B as output from RAMs U17E and U18E. This first RAM buffer line outputs to the weight bus under control of the F1-bar signal. Consequently, when the first line buffer is selected with the F1-bar signal to output the stored intensity onto the channel-II bus through multiplexers U16D and U17D, the weight addresses from RAMs U17E and U18E are applied to the weight bus through multiplexer U18B under control of the F1-bar signal.

### Kernel Logic

The kernel logic for spatial processing will now be discussed with reference to Fig 6AG. This kernel logic includes a 9-pixel kernel implemented with registers U19A to U19D, U20A to U20D, and U21A to U21D.

The 9-pixel kernel is composed of 3-adjacent buffered lines and 3- adjacent pixels on each line. The 3-lines are shifted out of the 3-line buffers into a 3-pixel register for each line buffer. Because each pixel is shown implemented as an 8-bit intensity word and because each register has 6 flip-flops, 1-1/3 register chips are needed to store a pixel word. Therefore, the 3-lines of 8-bit pixels are stored in 4-lines of 6-bit registers.

Referring to the pixel designation of Fig 5D, the following pixel storage is implemented with the kernel registers. Pixel-0 is stored in 6-bits of U19A and 2-bits of U19B. Pixel-1 is stored in 6-bits of U20A and 2-bits of U20B. Pixel-2 is stored in 6-bits of U21A and 2-bits of U21B. Pixel-7 is stored in 4-bits of U19B and 4-bits of U19C. Pixel-8 is stored in 4-bits of U20B and 4-bits of U20C. Pixel-3 is stored in 4-bits of U21B and 4-bits of U21C. Pixel-6 is stored in 2-bits of U19C and 6-bits of U19D. Pixel-5 is stored in 2-bits of U20C and 6-bits of U20D. Pixel-4 is stored in 2-bits of U21C and 6-bits of U21D.

The above assignment of 8-bit pixels to 6-bit registers can be seen with reference to the output lines at the left of Fig 6AG. These output lines show the 8-lines grouped together for each of the 9-kernel pixels and show the source registers for these lines.

The 3-lines of pixels are scanned out of the line buffers onto the channel-I bus, channel-II bus, and channel III bus; as shown at the left hand side of Figs 6Z, 6AB, 6AD, and 6AF. The 3-channel scanline pixels are each implemented with 8-pixel lines being input to registers U19A to U19D with the above described partitioning for triple 8-bit pixel bytes loaded into quadruple 6-bit registers. The pixel information in registers U19A to U19D is shifted into registers U20A to U20D respectively and is also output to the spatial filter logic as pixel-0, pixel-7, and pixel-6 respectively of the kernel. The pixel information in registers U20A to U20D is shifted into registers U21A to U21D respectively and is output to the spatial filter logic as pixel-1, pixel-8, and pixel-5 respectively of the kernel. The pixel information in registers U21A to U21D is output to the spatial filter logic as pixel-2, pixel-3, and pixel-4 respectively of the kernel. This shift register operation is performed under control of the CPE clock U8B-4 which clocks the information out of the line buffers, into the kernel registers, and through the kernel registers to provide a 9-pixel kernel in parallel to the spatial filter logic.

(c) DPL  
10/15/94

In a non-filtered implementation, the center pixel is provided to the video DACs on the rear end board to display the center pixel without spatial filtering. The center pixel is shown output on cable C-4 lines 22, 24, 26, 28, 30, 32, 34, and 36.

In a spatial filtering implementation, the output of the 9-pixel kernel is provided in parallel to the sum-of-the-products logic that performs spatial filtering. The output of the sum-of-the-products processing is provided to the video DACs on the

rear-end board to display the spatially filtered pixel. The 9-pixel kernel is shown output in parallel through the 72-lines (9-pixels by 8-bits per pixel) at the left hand side of Fig 6AG.

### Weight Logic

The weight logic will now be discussed with reference to Fig 6AH. The weight logic is implemented with registers U22A and U22B; decoders U22C and U22D; RAMs U21E to U23E; and multiplexers U23C and U23D.

The weight addresses stored in the line buffers are output to the weight RAMs U21E to U23E for accessing of the weights associated therewith. In one configuration, the weights to be accessed are associated with the center pixel in the kernel. Therefore, the weight addresses from the line buffer are shown delayed 2-clock periods with registers U22A and U22B so that they are available as addresses to the weight RAM when the pixel with which they are associated is in the center pixel position, pixel-8 position, of the kernel. Multiplexers U22C and U22D provide for addressing of weight RAM U21E to U23E from two sources, for reading and for writing.

During the run mode, addresses from the line buffers are selected with multiplexers U22C and U22D for accessing weights stored in weight RAMs U21E to U23E. During the load mode, the computer can generate weight addresses through the output port for writing weights into the weight RAMs; where the addresses selected with multiplexers U22C and U22D and are applied to weight RAMs U21E to U23E.

The RAM IO lines IO0 to IO3 are used for both input and output of data, controlled with the WE-bar pins. In the run mode, the W2-bar and W3-bar signals to the WE-bar pins are high, selecting reading from the weight RAM for outputting on the RAM IO lines to the spatial filter logic. In the load mode, the W2-

bar and W3-bar signals can be controlled by the computer to be low, selecting writing into the weight RAM from the computer output signals through multiplexers U23C and U23D. The weight addresses stored in register U22B are output to multiplexers U22C and U22D for accessing of weights from the weight RAM when selected by the run mode signal to multiplexers U22C and U22D. The weight addresses from the computer output port are applied to multiplexers U22C and U22D for storing of weights in the weight RAM when selected by the load mode signal to multiplexers U22C and U22D.

The weight bus communicates the weight address bits from the line buffers to weight register U22A; where the weight address is shifted into the weight registers to address the weight RAMs in a manner similar to the pixel intensities being shifted into the kernel registers for spatial filtering, as discussed above. The weight addresses are composed of the Y-weight address YW0B, YW1B, and YW2B and the X-weight address XW0B, XW1B, and XW2B shifted into register U22A.

Multiplexers U22C and U22D select the address for the weight RAMs, comprising the buffered weight addresses from register U22B in the run mode and the computer generated weight addresses XW0, XW1, XW2, YW0, YW1, and YW2 in the load mode when the computer is loading the weight RAMs. The multiplexer source information is selected with the load/run signal from the computer. The addresses from the multiplexers are used to address the weight RAMs to access the appropriate weights as a function of the weight addresses from the line buffers in the run mode and to

store the computer generated weights as a function of the computer generated weight addresses in the load mode when the W2-bar or W3-bar signals select writing into the weight RAMs. The data to be loaded into the weight RAMs in the load mode is input to tri state buffers U23C and U23D and is controlled to be applied to the RAM IO lines with control signal C2-34 to the buffer circuits. When the input data is applied to the RAM IO lines, the input data can be stored in the RAMs under control of the W2-bar signal or the W3-bar signal to RAMs U21E to U23E. When the input data is not applied to the RAM IO lines through U23C and U23D, the weights stored in the RAMs are output on the RAM IO lines to the spatial filter.

## REAR-END BOARD

The rear-end board interfaces the system to a CRT monitor and provides synchronization and clock signals for the CRT monitor and for the rest of the system. The rear-end board also performs auxiliary functions, such as converting analog joystick signals to digital form for control of a display processor. The rear-end board is shown in Figs 6S to 6V in detailed schematic diagram form.

A clock pulse generator 630A is implemented with a pair of inverters, an 18.432-MHz (herein referred to as 18-MHz for convenience) crystal, resistors, and capacitors as shown in Fig 6T to generate a square wave signal from inverter 630B pin-12. A counter circuit (74LS163N) is clocked from the inverted 18-MHz signal to pin-2 through inverter 74LS04 pin 2 for counting down the 18-MHz signal to about a 9-MHz clock signal for the display processor output from pin-14 and about a 2-MHz clock signal to the sync generator circuit MM532I from pin-12. A group of 4-switches with pull-up resistors are connected to the preload inputs of the counter on pins 3 to 6 to preload a selected amount for implementing count periods other than binary numbers.

A synchronization signal generator is implemented with a National Semiconductor MM532I component. The MM532I is connected in a usual fashion, such as described in the specification sheets and shown in Fig 6T. Switches D.C., V.R., and H.R. select MM532 modes of operation.

The MM532I horizontal drive signal is output from pin-15 and is used to blank the digital to analog converters (DACS) and is output through an 8T98-9 inverter H-DRIVE for generating the

horizontal sync signal or line sync signal CLS to the digital logic boards. The horizontal drive signal is also output to flip-flop U1-9, which is used to experiment with horizontal blanking HB signals.

The MM5321 vertical drive signal is output from pin-11 and is used to generate vertical blanking signals through an 8T97-5 to vertical blanking flip-flops U2, generating vertical blanking signal VB which is used to blank the video DACs. The vertical blanking signal VB U2-9 is buffered with 8T97-11 through cable C4-2 to apply the vertical blanking signal to the digital logic as the CFS signal.

The MM5321 composite sync signal is output from pin-16 and is used as the composite sync signal to the CRT monitor, buffered with an 8T97-3 and an 8T98-3 for complement signals.

The MM5321 blanking clock signal is output from pin-14 and is used to clock the vertical blanking flip-flops U2 and the horizontal blanking flip-flops U1-9 HB through an inverter 8T98-7.

The MM5321 interlace control signal is output from pin-9 and is used to control flip-flop U1-5 FLD-bar to generate the field-1-bar signal on cable C4-12 to the control logic board.

Joystick input circuits are shown in Figs 6U and 6V. Fig 6U provides the control circuits for the joystick analog to digital converters (ADCs) and Fig 6V shows the ADCs. These ADCs are input to the computer through the control logic board to provide operator control of display processing.

The joysticks utilized in this configuration are Apple-2 Compatible Joysticks named Computer Compatible Joystick. They

are analog joysticks having analog potentiometers for analog control. The joystick signals are input through plugs PJ1 and PJ2 at pin-6 and pin-7 for connection to the scaling amplifiers and ADCs shown in Fig 6V. The ADCs are controlled to start the conversion with the frame sync transition of the frame sync signal by shifting the frame sync signal through flip-flops U9A-5 and U9A-9 and detecting the condition of U9A-5 being in the 0-state and U9A-9 being in the 1-state with AND-gate U12-3 to generate a 1-clock period start convert pulse to the ADCs. The ADCs will start the conversion in response to this start convert pulse and will latch up the converted digital number for input to the computer under program control.

The two computer signals SEL0 and SEL1 are received from the computer through the control logic board to select one of four ADC numbers for input to the computer. These signals are inverted with U13-7 and U13-9. The inverted and non-inverted SEL0 and SEL1 signals are decoded with U11-3 for ADC 1-selection with signal S1, ADC-2 selection with signal S2, ADC-3 selection with signal S3, and ADC-4 selection with signal S4. Inverters U10 and U13 receive 8 signals from the ADCs, shown in Fig 6V, and buffers these 8-signals to drive the cable for input to the computer through the control logic board.

Consequently, the computer under program control generates select signals SEL0 and SEL1 and reads the 8-bit number from the selected ADC. In one implementation, the computer under program control generates four sequential select signal codes on lines SEL0 and SEL1 to address the four ADCs in sequence and inputs the selected ADCs output number for processing under program control.

The 4-ADCs and associated scaling amplifiers are shown in Fig 6V. The ADCs are implemented with the well known ADC0800 components, such as manufactured by National Semiconductor. The scaling amplifiers are implemented with the well known LF356 amplifiers. Each of the 4-joystick inputs from plugs PJ1 and PJ2 (Fig 6U) are shown connected to a different ADC channel through a scaling amplifier U1, U2, U5, and U6 (Fig 6V). The scaling amplifiers are connected in a conventional manner with resistor and capacitor networks to scale the joystick signals. Each scaling amplifier has its output on pin-6 connected to the input of its related ADC on pin-12. The ADCs convert the analog signal input at pin-12 in response to the start convert signal to pin-6 and latch the converted number in an internal register. The internal register is implemented with a tristate output controlled by the output enable signal to pin-7. The corresponding output lines of each ADC are connected together to form an 8-bit bus D0 to D7 which is routed to buffer amplifiers U10 and U13 (Fig 6U). Consequently, when one of the ADCs is tristate-enabled with one of the decoded select signals S1 to S4 (Fig 6U) input to pin-7 of the ADCs (Fig 6V), the number converted by that ADC is applied to the 8-bit data bus for communication to the computer.

One channel of video DAC is shown in Fig 6S. Each of the three channels are implemented with similar signals, except that the green channel having 3-bits is connected to data bits D5 to D7 and the red and blue channels having 2-bits are connected to data bits D6 and D7. These video DACs are high speed DACs for

converting intensity signals from digital signal form, as generated by the display processor, to analog signal form for exciting a CRT monitor. Three video DACs are used to convert three video signals; the red video signal, the blue video signal, and the green video signal; to generate the RGB signals to the CRT monitor. The video DACs can be implemented with the TDC1016 DACs manufactured by TRW. The connections for this DAC are shown in the VIDEO DAC CONNECTION TABLE provided herein. The D9 and D10 data pins are connected to ground. The digital red and blue signals having 2-bits resolution are connected to the D7 and D8 data pins. The digital green signal having 3-bits resolution is connected to the D6 and D8 data pins. The other data pins are connected to ground.

Buffer amplifiers are implemented with the well known LM359 buffer amplifier, where the buffer amplifier connections are shown in Fig 6S. The output of each buffer amplifier excites one of the red, green, or blue <sup>inputs</sup><sub>^</sub> to the CRT monitor.

VIDEO DAC CONNECTION TABLE

1	NC	NO CONNECTION
2	VEE	-5VDC
3	COMP	6.8 MICROFARAD CAPACITOR TO -5VDC
4	REF	-1V
5	AGND	GROUND
6	AGND	GROUND
7	OUT	VIDEO OUTPUT TO BUFFER AMPLIFIER
8	GND	GROUND
9	VCC	+5VDC
10	DGND	GROUND
11	<u>NDIS</u>	BLANKING SIGNAL
12	<u>CLK</u>	CLOCK
13	<u>CLK</u>	
14	<u>NDIS</u>	
15	D1	MOST SIGNIFICANT BIT, COMPLEMENT
16	D1	MOST SIGNIFICANT BIT, UNCOMPLEMENT
17	<u>N2C</u>	+5VDC
18	D2	DATA BIT 2, COMPLEMENT
19	D2	DATA BIT 2, UNCOMPLEMENT
20	NFH	GROUND
21	<u>NFL</u>	GROUND
22	D3	DATA BIT 3, COMPLEMENT <del>- GROUND</del>
23	D3	DATA BIT 3, UNCOMPLEMENT <del>- GROUND</del>
24	D4	DATA BIT 4, COMPLEMENT <del>- GROUND</del>
25	D4	DATA BIT 4, UNCOMPLEMENT <del>- GROUND</del>
26	D5	DATA BIT 5, COMPLEMENT <del>- GROUND</del>
27	D5	DATA BIT 5, UNCOMPLEMENT <del>- RED, BLUE GROUND; GREEN SIGNAL</del>
28	D6	DATA BIT 6, COMPLEMENT <del>- GROUND</del>
29	D6	DATA BIT 6, UNCOMPLEMENT <del>- RED, GREEN, BLUE SIGNAL</del>
30	<u>D7</u>	DATA BIT 7, COMPLEMENT <del>- GROUND</del>
31	D7	DATA BIT 7, UNCOMPLEMENT <del>- RED, GREEN, BLUE SIGNAL</del>
32	D8	DATA BIT 8, COMPLEMENT <del>- GROUND</del>
33	D8	DATA BIT 8, UNCOMPLEMENT <del>- GROUND</del>
34	D9	DATA BIT 9, COMPLEMENT <del>- GROUND</del>
35	D9	DATA BIT 9, UNCOMPLEMENT
36	D10	LEAST SIGNIFICANT BIT, COMPLEMENT
37	D10	LEAST SIGNIFICANT BIT, UNCOMPLEMENT
38	NC	NO CONNECTION
39	NC	NO CONNECTION
40	NC	NO CONNECTION

*MPW*  
10/15/84

## CIRCUIT SPECIFICATIONS

The circuits used in the experimental system are generally commercially available circuits that are well known and that are described in widely distributed specification sheets and component catalogs. A list of these specification sheets and catalogs is provided hereinafter and the materials referenced therein are incorporated herein by reference. For example, the 74LS00, 74ALS00, and 74AS00 specifications are set forth in the referenced Texas Instruments and Motorola catalogs and the Intel 8216 bus interface and the Intel 2149 RAM specifications are set forth in the referenced Intel catalogs; which are herein incorporated by reference.

1. Texas Instruments, ALS/AS Logic Circuits Data Book, 1983.
2. Texas Instruments, The TTL Data Book, Volume 3, 1984.
3. Texas Instruments, The TTL Pocket Data Book, 1983.
4. Intel, Component Data Catalog, 1981.
5. Intel, Memory Components Handbook, 1984.
6. Motorola, Schottky TTL Databook, 1981.

Various circuits used in the experimental system are described in the following list of component specifications, which are herein incorporated by reference.

1. TRW, LSI D/A Converters, TDC1016J-8/9/10.
2. TRW, Monolithic Video D/A Converters; TDC1016J-8, TDC0106J-9, TDC1016J-10; 1979.
3. Texas Instruments, TMS-4016, 2048-Word By 8-Bit Static RAM.
4. National Semiconductor, ADC0800 8-Bit A/D Converter.
5. National Semiconductor, MM5321 TV Camera Sync Generator.

6. Signetics, Hex Buffers/Inverters; 8T95, 96, 97, 98.
7. Mitsubishi; M58725P,S;P-15,S-15; 16384-BIT (2048-word by 8-bit static RAM.

PREPROCESSOR

### General Description

A preprocessor may be used in conjunction with the system of the present invention. The preprocessor can be placed <sup>inbetween</sup> the source of image information; such as the database memory, video camera, or other source; and the image memory. It can be used for preprocessing of the image prior to loading into image memory; such as for compression, filtering, shadowing, and other preprocessing. Compression may be needed to iteratively compress an image with anti-aliasing to overcome undersampling and aliasing that may be caused with the geometric processor processing the image in image memory. Filtering may be needed to reduce the spatial frequencies to overcome aliasing that may be caused with the geometric processor or the preprocessor compressing the image.

The image received from the source of the image can be buffered in a buffer memory for preprocessing. The buffered image can be loaded into image memory for geometric processing when the geometric processing goes beyond the limits of the image stored in image memory. For example, when the image in image memory is to be translated past a boundary of image memory, or is to be compressed below a compression threshold, or otherwise processed beyond the limits of the image; the buffered image can be loaded into image memory to continue geometric processing.

### Input Buffer Memory

An input buffer memory can be implemented to provide front-end buffering and to facilitate preprocessing. This input buffer memory can be a larger and slower memory than image memory because real time image processing may not be necessary from the input buffer memory. Alternately, this input buffer memory can be larger, smaller, or the same size as image memory and this input buffer memory can be faster, slower, or the same speed as image memory. Real time image processing can be provided out of image memory, as described herein. The input buffer memory can be used as an input buffer, such as for overlaying from a slower database memory, and can be used as an offline preprocessor memory; such as for compressing, filtering, decompacting, and shadow processing.

An input buffer memory capability can be provided between a slower speed database memory, such as a disk memory, and a higher speed image memory, such as a high speed RAM. The input buffer memory can be implemented with RAMs for interfacing the lower speed database memory and the higher speed image memory. Image information can be overlaid from database memory into the input buffer, such as in response to lookahead processing. Image information can be overlaid from the input buffer into the image memory, such as in response to geometric processing causing the window to approach an edge of image memory. The input buffer may have an access time that is 1000-times better than a database memory access time. Therefore, this may reduce the size requirement for the higher speed and hence higher cost image memory because the image memory then does not have to accommodate

longer access delays from database memory.

An input buffer memory can be used as an offline pre-processor memory. As discussed herein, compression, filtering, and decompaction (data decompression) can be performed before loading image memory, such as for high levels of compression and for an anti-aliasing low pass filter. A filter processor having kernel and sum of the products logic can be connected for operation with the input buffer. A compression processor, such as discussed herein with the geometric processor, can be connected for operation with the input buffer, as discussed herein for connection and operation with image memory. Also; shadow processing can be performed before loading image memory, such as by projecting a source of illumination onto topological altitude variations of the image and storing projected shadows in the pixel words. The image having filtering, compression, decompaction, and shadow processing can be overlayed into image memory to facilitate geometric processing.

### Progressive Compression

Compression using undersampling can be performed instantaneously. This is because the pixels for display can be selected directly with the address generator. Compression with anti-aliasing can be implemented with a filter kernel, such as a 9-pixel kernel; but may or may not be performed instantaneously, depending on the implementation.

Improved filtering can be obtained by compressing with a magnitude consistent with the filter kernel size and weights. For example, large compression, such as 5 to 1 compression, can exhibit undersampling characteristics even when filtered with a 9-pixel kernel. This is because the spatial period is significantly lower than the kernel width or period, such as a 5-pixel sample period and a 3-pixel kernel period. Improved smoothing can be obtained by maintaining the sampling period within the kernel period, such as a sampling period of 2-pixels or less for a 3-pixel kernel. This reduces undersampling, preserves image detail, reduces aliasing, and permits display of details reduced to subpixel size.

The combination of reduction of undersampling and obtaining of large compressions can be obtained with progressive compression. For example, assuming that a 9-pixel kernel permits compression by a factor, such as by a factor of 2, without undersampling; multiple compression iterations having a compression of 2-times or less can be performed to obtain compressions of greater than 2-times. For example, similar to the arrangement discussed with reference to the COMPRESSION PARAMETERS TABLE provided herein; a compression of 32-times can

be obtained by 5-iterations of compression by a factor of 2; where the first iteration reduces size to 1/2, the second iteration reduces size by another factor of 1/2 to 1/4, the third iteration reduces size by another factor of 1/2 to 1/8, the fourth iteration reduces size by another factor of 1/2 to 1/16, and the fifth iteration reduces size by another factor of 1/2 to 1/32. Therefore, progressive compression provides an exponential type of compression, i.e., 32 equals 2 exponent 5.

Progressive compression yields efficiencies in addition to the above exponential relationship for progressive compression. For example, as an image is compressed, the number of pixels in that image is reduced. If the compression processor can process a number of pixels per second, further compression can proceed at an increased rate. For example, in the above progressive compression example, a 512-pixel by 512-pixel image (265,000 pixels) compressed to a 1/32 size is compressed to about 100-pixels by 100-pixels (10,000 pixels). Each progressive compression iteration proceeds 4-times as fast because it contains only 1/4 of the number of pixels (1/2 size squared equals 1/4 pixels). For example, the first iteration compresses the image from 512-pixels by 512-pixels to 256-pixels by 256-pixels in one compression iteration. The second iteration compresses the 256-pixels by 256-pixels to 128-pixels by 128-pixels in 1/4 of a compression iteration because there are only 1/4 of the number of pixels to be compressed in this progressively compressed image. Therefore, the exponential nature of progressive compression (i.e., 5-iterations of 1/2-

compression for a 32-times compression) is further exponentially improvement in compression time as the image becomes more compressed. Consequently, progressive compressions of even high compression amounts and even with relatively small kernel sizes will progress rapidly.

### Outer Loop Correction

Progressive image processing on a memory map image can cause propagation of errors. For example, progressive image processing, such as compression, can involve progressive updating of pixel information in image memory with a resultant buildup of errors due to the progressive updating. The error buildup can be bounded with an outer loop correction that re-establishes initial conditions. For example, after a predetermined number of image processing iterations; the initial condition image frames can be reloaded from the database and then updated to the present conditions.

An example of an outer loop correction will now be provided to better illustrate an implementation. As discussed herein, a plurality of mosaic images can be loaded from database into image memory and can be progressively processed in image memory to provide a progressively updated image. As the image is progressively updated, errors tend to be cumulative. The errors may be in the form of spatial positional errors, intensity errors, and other errors. An outer loop correction can be implemented by reloading mosaics from the database into image memory, thereby re-initializing image memory and effectively nulling out accumulated errors. The new initial conditions in image memory can then be updated to the image conditions that existed immediately prior to reloading from the database to provide the same image conditions but without an accumulated error.

Updating of image memory can be controlled in various ways; such as based upon adaptive determinations, pre-determination, or

other determinations. Adaptive determinations can be implemented by adaptively monitoring features of the image to establish when the error buildup exceeds a threshold. For example, a reference image covering several pixels can be stored in image memory and can be monitored for smearing into other pixels or can be monitored for positional deviations. An outer loop correction can be initiated when the smearing or positional deviations exceeds a threshold. Pre-determination can be implemented by updating area memory after a pre-determined number of updates; such as after 1,000 image memory updates.

Re-initializing of image memory can place initial conditions into image memory that may be different from the updated conditions that are being error bounded. Therefore, the initial conditions may be processed to the updated conditions prior to re-initialization, such as in an input buffer memory, or after re-initialization, such as in image memory.

After re-initializing of image memory, subsequent updating may again introduce error accumulation and subsequent re-initialization may again be used to bound error accumulation.

ADDITIONAL FEATURES

### Scenario-Related Detail

One form of enhancing a moving display; such as a moving map display, a simulation display, or other display; is to expand the image as the range to the image is reduced for greater detail and reduced area and to compress the image as the range to the image is increased for reduced detail and greater area. Detail can also be controlled as a function of velocity and other parameters, as discussed below.

Increasing of velocity can increase the rate that the image is scrolled, to be consistent with the increased motion. Increasing the scrolling rate can increase the rate that the database memory is accessed to support this increased scrolling rate. The latency time associated with the database memory can limit maximum speed that can be supported with this system. However, maximum speed can be significantly increased based upon using a speed variable detail implementation.

As relative speed increases, the amount of detail that an observer can process decreases. For example, a low flying aircraft at high speed covers ground rapidly, but the ground appears as a "ground rush" that reduces the pilot's ability to perceive details. Therefore, it may not be necessary to provide high detail information for conditions where vehicle velocity prevents perception of these details. Implementation of such a rationale can increase maximum scrolling velocity of an image. For example, "ground rush" can be implemented with a low detail image that is interpolated, such as with spatial processing. In accordance with the discussions herein on storing of images having different amounts of compression and the discussions

herein on expanding of images, lower detail images can be provided and can be expanded to encompass larger areas, either with or without spatial filtering, to provide a larger image at lower detail. Use of an expanded lower detail image, with or without spatial filtering, can provide larger images with fewer pixels. Such images can be scrolled with lower traffic from the database memory; permitting slower database memories and greater scrolling velocities.

## Overlays

Overlays can include combinations of processed images and graphic polygons. For example, in a map display; processed images can be used for background terrain and for certain objects overlayed thereon and graphic images can be used for symbols and polygon images overlayed thereon.

Overlays can be implemented in various ways. Images can be overlayed together in the same image plane; where occulting is then fixed and motion therebetween is then fixed. Images can be overlayed in different planes of the same image processor; where occulting is selectable, such in accordance with occulting priorities assigned to each plane, and motion therebetween is fixed. Images can be overlayed in different planes of different image processors; where occulting is selectable, such in accordance with occulting priorities assigned to each plane, and motion therebetween is controllable, such as in accordance with independent image processors controlling different images. Image overlays and graphic overlays can be mixed together; such as in the same image plane, in different image planes of the same image processor, and in different image planes of different image processors.

Graphic overlays can be adapted for different applications. Cursors, crosshairs, sights, and other such graphic symbols can be used to identify selected portions of images. Alphanumerics can be used for annotating images. Graphic polygons can be used for overlay images of lower detail than processed images. CIG 3D images can be used for more complex overlay images of lower detail than processed images.

Overlays can include pictorial features, annotation symbols, imaginary pathways for map displays, and other such overlays. In a military application; overlays can be related to fire control, bombing, sensors, and navigation. Navigation information can (Global Positioning System) include GPS, inertial, celestial, radar, Tercom, dead-reckoning, and other navigation information. Sensor information can include radar, infra-red, video, sonar, and other sensor information.

A large number of textured overlays can be provided. Each textured overlay can be independently processed for geometric operations including rotation, translation, scaling, 3D-perspective, and warping; cropped to irregular external and even internal features; and overlaid with a background image and with other overlay images using occulting priorities on a pixel-by-pixel basis. This provides high detail occulting between irregular textured images for high detail interaction in real time.

Textured overlays can be cropped to irregular external and internal features; i.e., a tree has an external outline defined by the external leaf and branch outline and has internal spaces between leaves and branches. Textured overlaying can be performed to pixel resolution and pixel detail based upon occulting priorities. Therefore, a tree image can be oriented, positioned, scaled, warped and overlaid on a background image; occulting the background image, occulting portions of more remote objects, and being occulted by portions of nearer objects. Moving occulted features can be seen between the cropped portions of occulting overlays; i.e., a partially occulted tank image can be seen between the leaves of an occulting tree image as it moves behind the tree image and can be seen as it moves past the external outline of the tree image. Occulting can be performed on a pixel-by-pixel basis, so that a leaf or branch can occult a feature of a more remote object in one pixel and an adjacent space between the leaves and branches can display a non-occulted feature of the same more remote object in an adjacent pixel.

Graphic overlays can be implemented in various forms with the system of the present invention. A graphic overlay can be geometrically processed, such as being contained in pixel words associated with an image that is being geometrically processed or such as being contained in pixel words dedicated to the graphic overlay. Alternately, graphic overlays can be fixed, such as being fixed relative to the viewport. Geometrically processed graphic overlays can be rotated, translated, expanded, compressed, warped, and otherwise geometrically processed either together with an image related thereto or separately, independent of other images.

Graphic overlays that are associated with images and geometrically processed with such associated images will now be discussed. Graphic overlays can be written over an image in image memory to provide a permanent overlay. Such permanent overlays have been demonstrated with the BASIC PROGRAM LISTING LD.ASC Basic provided herein by loading graphic vectors, rectangles, and test patterns and by loading textured image test patterns over an image in image memory. Such permanent overlays effectively erase the previous image information in the pixels that are overlaid, making it more difficult to change the overlay. Graphic overlays can be applied in a manner that permits convenient changes by storing the graphic overlay in extra bit positions in the pixel words. Hence, the overlay does not erase the previous image information; where both, the previous image and the graphic overlay are stored separately in the pixel words. Selection logic can be used to select the graphic overlay if an overlay intensity is stored in the pixel

word for overlaying the image and to select the image if a graphic overlay intensity is not stored in the pixel word. Overlay bits in a pixel that are all 0-set can be indicative of a condition where a graphic overlay intensity is not stored in the pixel word, <sup>causing</sup> <sub>^</sub> the image bits to be displayed. Graphic overlay bits in a pixel that are not all 0-set can be indicative of a condition where a graphic overlay is stored in the pixel word, causing the graphic overlay in that pixel to be displayed in place of the image bits. An OR-gate can be used to detect non-zero overlay bits for selecting the overlay or for selecting the image bits in response to non-zero overlay bits or in response to zero overlay bits, respectively.

Graphic overlays that are independent of textured images and that are processed independently of other images will now be discussed. Graphic overlays can be written into a separate graphic image memory. This graphic image can be geometrically processed, as discussed for processing of textured images in image memory. The geometrically processed image can be output for display, can be overlaid on other graphic images and processed images in other image memories, and can be otherwise processed as discussed for processing of textured images herein.

Graphic overlays that are fixed in the viewport will now be discussed. Graphic overlays can be written into a separate graphic image memory. These graphic overlays can be referenced to the <sup>view</sup><sub>^</sub>port and can be fixed to the viewport; such as with a viewport frame, alphanumeric characters, and other features that are to remain stationary in the viewport. This fixed graphic

image can be scanned-out of the related image memory in raster scan form without rotation, translation, expansion, compression, or other geometric processing.

Overlays between textured images that are geometrically processed independently of each other will now be discussed.

A textured overlay approach will now be described where the background image is kept separate from the overlay images. The overlay images are stored in separate image memories. The background and overlay image memories can then be simultaneously scanned out to the display. The background image memory can be implemented in the form previously discussed for a textured image being scanned out with rotation, translation, compression, and expansion image processing. The overlay image memory can similarly be scanned-out.

Switching between two scanned-out pixel streams can be implemented as follows. Presence of an overlay image feature to be overlaid on the background image can be detected with various techniques, such as discussed below. The overlay image memory can have zero intensity for pixels that do not have overlay image features therein and that the background is to be provided therein. Alternately, a flag can be included in the pixel word and can be used to identify pixels having overlay features therein. Alternately, an overlay code (i.e., one-code out of 256-codes for an 8-bit pixel) can be used only for non-overlaid pixels, where overlaying of pixels can use other codes except for the overlay code. The overlay image can be selected if an overlay image feature is not in that pixel and the background image is selected if an overlay is in that pixel. The selected image can

be routed through the DACs to the display monitor.

Overlay images, exemplary of multiple combined images, can be dynamically manipulated, as with the background image. Each overlay generator can have its own image memory and address generator logic. Outputting of the overlays can be keyed to occult a background image and other overlay images.

Multiple images can be overlaid using the experimental system configuration. One arrangement is to replicate the image processing portions of the system (i.e., BL1, BM1, and BM2) for generating a plurality of parallel RGB video signals from the different image processing channels. Each channel can be assigned a different occulting priority. The background image can be assigned the lowest occulting priority. Overlay images from overlay channels can be assigned higher priorities for overlaying on the background image. The images, including the background image and a plurality of overlaying images, can be individually processed in accordance of a scenario for the particular image. For example, each of the images can be individually translated, rotated, expanded, compressed, and warped during scanout and can be combined with a combiner circuit to provide the background image with a sequence of levels of occulting overlay images superimposed thereon for output to the CRT interface under control of the combining circuit.

An alternate overlay arrangement simultaneously scans a background image into a refresh memory; then scans overlay images, such as in occulting sequence, into the refresh memory; and then refreshes a display from the refresh memory. A multiport

refresh memory configuration permits loading in background and overlay information simultaneously. A double buffer refresh memory configuration permits loading of one refresh memory while refreshing a display with another refresh memory.

An image to be overlaid on a background need not be a regular shaped image, such as a rectangular image; but may be an irregular shaped image, such as an aircraft or tree image. It may be desirable to overlay an irregular image on a background image, including the textured pixels within the irregular outline, and without overlaying the pixels from the overlay memory map outside of the irregular image on the background image. One configuration for overlaying of pixels within an irregular outline and not overlaying pixels outside of the irregular outline can be implemented by identifying the pixels inside of the irregular outline in contrast to pixels outside of the irregular outline. This can be accomplished in various ways. Flags can be set or reset in pixel words to identify whether the pixels are outside of the overlayable pixel images. Alternately, an overlay can be implemented with a known background color, where the known background color identifies the cropped portions of the overlay to identify whether the pixels are outside of the overlayable pixel images. Detection of a pixel flag or a pixel color, indicative of a cropped portion of the image, can be used to disable overlaying of that image. Detection of a pixel flag or a pixel color, indicative of a non-cropped portion of the image, can be used to enable overlaying of that image consistent with occulting priorities.

Overlaying will now be discussed for the color detection implementation. A null color, such as black, can be loaded into the overlay memory and the overlay image can be overlaid thereon. Hence, pixels outside of the image would have a null color code, such as black. An inside condition can be identified by a null color code in a pixel word. Also, an outside condition can be identified by a null color code in a pixel word. As the pixel words are scanned out to the combining logic, detection of a color code is indicative of a pixel that is to overlay the background image and detection of a null color code is indicative of a pixel that is not to overlay the background. In this manner, overlaying images of both regular and irregular shape in an overlay memory can be overlaid onto a background image.

Overlaying will now be discussed for the pixel flag implementation. An inside condition can be identified by 0-setting a flag in each pixel word having an image feature on the outline and inside of the outline. Also, an outside condition can be identified by 1-setting a flag in each pixel word that is outside of the outline. As the pixel words are scanned out to the combining logic, detection of a 0-set pixel flag is indicative of a pixel that is to overlay the background image and a 1-set pixel flag is indicative of a pixel that is not to overlay the background. In this manner, overlaying images of both regular and irregular shape in an overlay memory can be overlaid onto a background image.

The combining circuit can be a series of tristate multiplexers for overlaying, such as 74LS365 multiplexer circuits, for selecting the appropriate pixel from a plurality of pixels from the plurality of channels. Selection logic can include detection of a flag or a color in the pixel word of each of the channels and a determination of which of the channels is to be enabled for display out of the plurality of channels for that particular pixel based upon image detection and image priority.

For example, a flag in each pixel word associated with each overlaying image can be 0-set if the overlaying image occupies that pixel and can be 1-set if the overlaying image does not occupy that pixel. Therefore, a plurality of overlaying images can be provided that are stored in a plurality of overlaying image memories where the pixels in each of the overlaying image memories that contain pixel information to be overlaid have a 0-set flag and the pixels in each of the overlaying image memories that do not contain pixel information to be overlaid have a 1-set pixel flag.

For convenience of discussion, an implementation may be discussed relative to overlaying of an overlay image on a background image. However, this is intended to be illustrative of overlaying of a plurality of images therebetween and on a background image. For example, priorities can be identified, such as with a priority register identifying the occulting priorities of the different overlay images. If every one of the overlay images for a particular pixel has that pixel cropped, then the background having the lowest occulting priority will be

displayed for that pixel position. If every one of the overlay images except for one overlay image for a particular pixel has the image cropped, then the single overlay image that is not cropped will be displayed, occulting the background image for that pixel position. If more than one of the overlay images for a particular pixel are not cropped, then the one of the overlay images having the greatest occulting priority will have that pixel displayed, occulting the background and other non-cropped images and not being occulted by higher priority cropped images for that pixel.

It may be desired to have a relatively large background image, such as 512-by-512 pixels or 1024-by-1024 pixels, and relatively small overlaying images, such as 64-by-64 pixels or 128-by-150 pixels. One method is to implement all image memories having the same size, such as 512-by-512 pixels. However, this may be an inefficient implementation if the overlay images are significantly smaller. Nevertheless, it may be advantageous to have address generators that operate over the full viewport environment, even for relatively small overlays. Both of these considerations can be satisfied, where an overlay image memory can be implemented in small size consistent with the dimensions of the overlay image, such as 64-by-64 pixels, and where the overlay image address generators can process the full region of the viewport, such as 512-by-512 pixels. This can be achieved by implementing the address generators to cover the full image memory space, such as with overlay address generators having the same range as the background image address generators; but only

populating the overlay image memories with a small portion of the address decode and RAM chips of the full range of the address generators. This small portion can be a portion large enough to store the overlay image, which may be significantly smaller than the background image. The overlay address generators can scanout a full image over the range of the background image, most of which may address pixel positions that are not populated. The default condition for a non-implemented pixel can be configured to be the same as for a non-overlay flag, such as a 1-flag. For example, a non-populated memory position can be represented by an open circuit. Therefore, pull-up resistors on output lines will automatically generate a 1-flag for a non-populated memory condition, indicative of a non-overlay flag and indicative of a cropped portion of the image. In this manner, relatively small dimension image memories can be used for relatively small overlay images to be superimposed on a relatively large background image, preserving simplicity of implementation of a plurality of address generators related to a plurality of overlayable images having dimensions consistent with the background dimensions for simplicity of registration and overlaying.

Each of a plurality of overlays can be manipulated in accordance with the single image implementation discussed above for translating, rotating, and compressing/expanding about a center coordinate and determining the initial point for the raster scan therefrom. Each of the overlaying images can be rotated, translated, and expanded/compressed to facilitate relative motion therebetween and relative to the background image independent of each other.

Each overlay image and the background image can then be translated relative to the viewport and, hence, positioned relative to each other. This facilitates ~~separate~~<sup>A</sup> controls for each of the images, such as ~~separate~~<sup>A</sup> joystick controls and ~~separate~~<sup>A</sup> digital computer controls, to permit continuous movement of each overlay image and the background image relative to each other.

The combining logic can generate the combined pixels directly to the CRT interface, such as to the DACs, or alternately can load the combined pixels into the buffer memory in accordance with the alternate configurations disclosed for the demonstration system, or alternately can load the combined pixels into an image memory or a refresh memory.

In a configuration having a single overlay image in combination with a background image, priority is implicit in the overlay, where an overlay flag or overlay color in a pixel word commands overlaying of the overlay image on the background image. In a configuration having a plurality of overlay images in combination with a background image, priority between the overlay images can be established in various ways, such as to provide control of occulting therebetween.

In a flag controlled occulting configuration, a plurality of flag bits can be implemented for each pixel word to encode priorities between overlays. In such a configuration, a 2-bit flag permits determining priorities between 3-overlay image pixels on a background image pixel. Similarly, a 3-bit flag permits determining priorities between 7-overlay image pixels on

a background image pixel.

In a wired priority configuration, hardwiring selects channel priorities relative therebetween by wiring channels to input boards, with selection of jumpers, and with other wiring techniques.

In a programmable priority configuration, selection of channel priorities relative therebetween is performed by a programmable priority word that controls the priority combining logic.

Changes in priorities, such as an occulting overlay becoming an occulted overlay, is facilitated with a multiple bit flag word arrangement and with a programmable priority word arrangement. Improved image memory efficiency is facilitated with a single bit flag in image memory or an overlay color detector, such as provided with the wired arrangement and the programmable word arrangement. Consequently, the programmable word arrangement provides the combination of advantages of programmable changes in priority together with improved image memory efficiency.

Modular expansion capability can be provided with a basic system having a background image that permits modularly adding one or more overlay image channels to meet the particular system requirements. This can be achieved by placing the multiplexing logic for each channel in the channel module and having a programmable priority selection arrangement in the standard module for generating selection signals to each of the plurality of channels. Each overlay channel can be implemented with integrated circuits on one or more printed circuit boards; can be implemented with a custom integrated circuit chip or chips; and

can be implemented with other modular configurations.

Operation of the experimental system shows that images can be overlayed in image memory, such as with the BASIC PROGRAM LISTING LD.ASC provided herein, and the composite overlayed image can be manipulated under joystick control. This is representative of dynamic manipulation of individual overlay images and a background image and then overlaying the manipulated overlays on the other overlays and background images.

Overlaying in the experimental system using the BASIC PROGRAM LISTING LD.ASC provided herein provides precise transitions between the overlaying image and the overlayed image without a fictitious border effect. This configuration permits individual cropping of an image, such as with an irregular shaped image; compressing a cropped image to a desired size; rotating a cropped image to the desired orientation, and translating a cropped image to the desired position. Continuous rotation, compression, expansion, and translation can be controlled; such as with operator joystick commands and computer commands; to provide continuous dynamic cropped overlays moving relative to each other and relative to the background. Also, continuous cropping can be provided, such by changing the flag or color structure of the overlay images.

Dynamic image effects can be provided by enabling and disabling overlays, such as under operator or computer control. For example, a running person image can be implemented with a plurality of different images, where each image is in a different state of a sequence of running motions. Selecting a sequence of

running overlays can provide the appearance of a running image.

Overlaying of images, as discussed above, is illustrative of other image interaction processing. For example, different images can be arithmetically combined, logically combined, and otherwise combined in addition to overlaying. Arithmetic combination includes adding, subtracting, multiplying, dividing, and other arithmetically combining intensities. Logical combination includes ANDing, ORing, XORing, AND/ORing, and other logically combining intensities. Because pixels are scanned out sequentially; combining intensities; such as overlaying, arithmetically combining, and logically combining intensities; can be performed on a pixel pair by pixel pair basis. Consequently, such combining can be implemented with a single one of each of the different combining circuits; such as a single set of adder, subtracter, multiplier, divider, and logical arrangements. Each pixel can have a single monochromatic intensity or a plurality of color intensities. For a monochromatic intensity, a single channel of pixel combining logic may be sufficient. For a color intensity arrangement, three channels of pixel combining logic may be needed for red, green, and blue intensities.

Overlays can be dynamically moving video images, such as real time video images from a video camera or video disk, and need not be limited to static overlays, such as from a database memory, that are dynamically manipulated. For example, in an animation type application; a running person can be generated with a plurality of images generated in sequence having different stages of motion. These video sequences can be a stored sequence, such as 32-frames of the overlay on a video disk showing a full running cycle that are superimposed in sequence.

F Overlaying may be implemented as selection and replacement, as previously discussed. However, for some applications it may be desirable to use addition. For example, two of the overlay planes can be used as full planes and operation between these two planes can be selected to add or to subtract from each other to give a sum or difference output image.

The overlaying implementations discussed above provide the advantages of being able to crop irregularly outlined objects and also to be able to crop internal to these outlines. This can be accomplished by setting of overlay flags. Overlay flags can be set and reset internal to crop an irregular outline as well as external to an irregular outline.

The above overlaying approach permits progressive spatial fading-in and fading-out of images. For example, setting of overlay flags permits dynamically spatially fading-in.

Multiple textured images can be superimposed or overlayed on the background image using image processing techniques discussed herein. For example, in an aircraft navigational simulation, a terrain image can be provided in a first image memory to be processed to translate, rotate, compress, and expand the terrain image to simulate motion over terrain. An aircraft image can be provided in a second image memory to be processed to translate, rotate, compress, and expand the aircraft image. The image processing of the aircraft image can be separate from the image processing of the background image to provide relative effects between the background image and the aircraft image. The aircraft image can then be overlayed on the background image to provide relative motion therebetween. Therefore, an observer can view the aircraft image moving relative to the background image, where both the aircraft image and the background image can be moving and can be moving relative to each other.

Overlaying of different images can be performed in various ways. One arrangement writes the overlaying image into the corresponding pixels of the overlayed image for a composite overlaying and overlayed image. Another arrangement maintains the overlaying image and the overlayed image in separate image memories and switching between the different image memories as a function of the ranges of the images (where range can establish occulting priorities) to occult the more remote overlayed images with the less remote overlaying images.

Different images can be moved, such as in rotation and translation, relative to each other and can be computationally connected together. For example, simulated human arm motion can be made up of motions of different arm elements relative to each other. As the shoulder moves, the upper arm, lower arm, and hand move accordingly. In addition, the upper arm can move relative to the shoulder and, as the upper arm moves, the lower arm and hand move accordingly. In addition, the lower arm can move relative to the upper arm and, as the lower arm moves, the hand moves accordingly. In addition, the hand can move relative to the upper arm and objects held in the hand move accordingly. Such action can be illustrated with an action game having a person swinging a sword.

An arrangement for implementing a game with an arm will now be discussed with reference to Fig 8A. Motion of shoulder 820A drives upper arm 820C, lower arm 820E, and hand/sword 820G/820H; motion of upper arm 820C relative to shoulder 820A drives lower arm 820E and hand/sword 820G/820H; motion of lower arm 820E relative to upper arm 820C drives hand/sword 820G/820H; and motion of hand/sword 820G/820H can be provided relative to lower arm 820E. Each of these elements; shoulder 820A, upper arm 820C, lower arm 820E, and hand/sword 820G/820H; can be implemented separately in separate image memories. Each of these elements can be rotated relative to the point of rotation, which is the shoulder joint point of rotation 820B for upper arm 820C, the elbow point of rotation 820D for lower arm 820E, and the wrist point of rotation 820F for hand 820G. Each element, in its separate image memory, can be rotated about the point of

rotation. The elements can then be connected together by overlaying each subsequent element in a manner that connects the point of rotation in the driven element to the corresponding point of rotation in the driving element. For example, upper arm 820C can be overlayed on shoulder 820A in a manner that places shoulder joint point of rotation 820B at the same image position for both, shoulder 820A and upper arm 820C. Also, lower arm 820E can be overlayed on upper arm 820C in a manner that places the elbow point of rotation 820D at the same image position for both, lower arm 820E and upper arm 820C. Also, hand 820G can be overlayed on lower arm 820E in a manner that places the wrist point of rotation 820F at the same image position for both, hand 820G and the lower arm 820E.

F  
Overlaying of the corresponding point of rotation for two elements can be provided by identifying the pixel that corresponds to the common point of rotation for both elements, the driving element and the driven element. As the driving element moves, the point of rotation moves correspondingly. It is desirable to keep track of the point of rotation as it moves. This can be achieved in various ways. One way is to identify the point of rotation with a unique code stored in a flag field of the pixel word. Another way is to compute the new position of the point of rotation from the old position of the point of rotation and the amount of motion.

A driven element can be overlayed on a driving element, i.e. hand 820G can be overlayed on lower arm 820E, by moving hand 820G so that the wrist point of rotation 820F of the hand image

overlays the wrist point of rotation for the lower arm 820E. As discussed above, this can be achieved by identifying the pixel in which the point of rotation of the driving element with an X and Y position and then translating the driven element so that the point of rotation of the driven element is overlayed on the same X and Y position of the point of rotation of the driving element.

The above relative motion of attached elements facilitates compound motion of the elements. For example, translation and rotation of shoulder 820A correspondingly translates and rotates upper arm 820C, lower arm 820E, and hand 820G; translation and rotation of upper arm 820C relative to shoulder 820A correspondingly translates and rotates lower arm 820E and hand 820G; and translation and rotation of lower arm 820E relative to upper arm 820C correspondingly translates and rotates hand 820G. Therefore, motion of upper arm 820C is the combination of motion of shoulder 820A and motion of upper arm 820C; motion of lower arm 820E is a combination of motion of shoulder 820A, motion of upper arm 820C, and motion of lower arm 820E; and motion of hand 820G is a combination of motion of shoulder 820A, motion of upper arm 820C, motion of lower arm 820E, and motion of hand 820G.

DATABASE MEMORY

### General Description

Various applications of the present invention involve scrolling over a large image, such as a large area of terrain. This invention is well adapted for scrolling over a large database image using a "virtual scrolling" feature. A database memory for storing a large database image can be readily implemented with available memories, such as disk memories. A review of disk memories for a large image database is provided below.

The database arrangements disclosed herein can include a memory management capability, such as with a relational database or an associative database, for conveniently accessing information for the operator. Alternately, the memory management capability can have a search capability for locating particular information within the database. Other memory management approaches can be used.

Two generic types of very large database memories are available, digital disk memories and analog disk memories. Digital disk memories are characterized by Winchester disk memories. Analog disk memories are characterized by optical video disk memories. Video disk memories have the lowest cost potential and greatest storage capacity potential. Large Winchester disks are readily available at an affordable price, are relatively easy to interface to, and have acceptable access rates. Therefore, large Winchester disks can be used for current configurations and optical disks can be used for longer range requirements. However, database memory can be implemented with any of the memory types discussed herein.

Large rotating memories exceeding 1-gigabyte capacity are currently available. Winchester disk memories exceeding 100-megabyte capacity are readily available at low cost. Assuming 1-byte per pixel image color capability, as provided in the experimental system discussed herein; an image can be provided having 10,000-pixels on a side with a 100-megabyte disk. Assuming a terrain resolution of 2-meters per pixel, this relates to a 20-kilometer square environment capability with a 100-megabyte disk memory. Use of larger disk memories, use of multiple disk memories in parallel, and use of image compression, permits this capacity to be significantly increased.

F Images can be implemented in different ways, such as by storing an image or by constructing an image from image primitive. For example, for a real world scene that must accurately represent a real world location, the actual image details can be stored in the database and can be accessed to display the real world environment. Also, for scenes that need not represent a real world location, image primitives can be stored in the database and can be accessed and overlayed to display the non-real world environment.

Images can be stored in the database in compressed or in non-compressed form. Storage in compressed form reduces the memory requirements and involves de-compression to restore the image. Storage in non-compressed form requires greater storage capacity, but does not need de-compression. For example, compressed image storage can be used for applications having significant database constraints, such as avionic applications,

and non-compressed image storage can be used for applications without significant database constraints, such as ground based simulation applications. Images implemented with either compressed or non-compressed storage can benefit from spatial filtering image enhancement capability to increase spatial resolution and to increase color latitude for smoothness and for anti-aliasing.

In non-real world applications, an unlimited amount of terrain can be constructed with a relatively small amount of database memory by overlaying terrain primitives stored in the database. The primitives can include forested terrain, desert terrain, hilly terrain, mountainous terrain, cultural features, graphics, alphanumerics, icons, and other such primitives. Terrain primitives can be combined to form a virtually unlimited number of simulated environments to construct enormous areas of terrain with a relatively small amount of database memory capacity. Terrain images can be constructed by overlaying background color, overlayed features, and details; such as a varying textured green and brown background having foliage overlays and cultural overlays to construct a simulated scene.

A database memory for an image processing system can be implemented with a fixed memory map having a stored image. The memory map can be implemented with RAMs, ROMs, or other such memory devices. A RAM memory map permits the loading of new information, such as over a data link from a remote database. A ROM memory map provides non-volatile images which can be changed with plug-in cartridges. As shown in Fig 2B, memory map 205B can be implemented to be larger than window 205A; where window 205A

can rotate, translate, and otherwise process memory map information for display. Use of a memory map without a large database image can significantly reduce system cost and can provide a dedicated system.

### Database Memory Implementations

Database memories can be implemented in various forms. These forms can be characterized as rotating memories and solid state memories, analog and digital memories, magnetic and non-magnetic memories, and other forms of memories. Many types of database memories have been described. For example, magnetic disk and optical disk memories have been described herein and charged coupled device (CCD) and bubble memories have been described in patent applications S/N 520,277; S/N 889,301; S/N 844,765; S/N 812,285; and other patent applications referenced therein and patents issuing thereon.

A digital magnetic disk memory, such as a Winchester disk memory, can be used as an image database memory. The digital magnetic disk memory can provide large capacity storage, such as hundreds of millions of bytes of storage per disk to billions of bytes of storage per disk. Such disks are manufactured by Ampex, Kennedy, Fujitsu, CDC, IBM, and other companies. A digital magnetic drum memory can be used as an image database memory, as with the digital magnetic disk memory. The digital magnetic drum memory can provide large capacity storage, such as hundreds millions of bytes of storage per disk to billions of bytes of storage per drum. Such drums are manufactured by CDC, IBM, Univac and other companies.

A digital magnetic tape memory can be used as an image database memory. The digital magnetic tape memory can provide large capacity storage, such as hundreds of millions of bytes of storage per tape. Such tape memories are manufactured by Ampex, Kennedy, CDC, IBM, and other companies. A digital magnetic tape

memory can be implemented in the form of a cassette, a reel, an endless loop, or other form.

An analog magnetic disk memory, can be used as an image database memory. The analog magnetic disk memory can provide large capacity storage.

An analog magnetic tape memory can be used as an image database memory. The analog magnetic tape memory can provide large capacity storage, such as hundreds of millions of bytes of storage per tape. An analog magnetic tape memory can be implemented in the form of a cassette, a reel, an endless loop, or other form.

An analog optical disk memory, such as a video disk memory, can be used as an image database memory. The analog optical disk memory can provide large capacity storage, such as billions of bytes of storage per optical disk. Such disk memories are manufactured by Pioneer, RCA, and other companies.

A digital optical disk memory can be used as an image database memory. The digital optical disk memory can provide large capacity storage, such as hundreds of millions of bytes of storage per disk to billions of bytes of storage per disk.

A digital CCD alterable memory can be used as an image database memory. The digital CCD alterable memory can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Such CCD alterable memories were manufactured by Intel and other companies. Digital CCD alterable memories are described in the previously referenced patent applications.

An analog CCD alterable memory can be used as an image database memory. The analog CCD alterable memory can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Analog CCD alterable memories are described in the previously referenced patent applications.

A digital CCD ROM can be used as an image database memory. The digital CCD ROM can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Digital CCD memories are described in the previously referenced patent applications.

An analog CCD ROM can be used as an image database memory. The analog CCD ROM can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Analog CCD ROMs are described in the previously referenced patent applications.

A digital magnetic bubble memory can be used as an image database memory. The digital bubble memory can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Such bubble memories were manufactured by Intel, IBM, and other companies. Digital bubble memories are described in the previously referenced patent applications.

An analog bubble memory can be used as an image database memory. The analog bubble memory can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Analog bubble memories are described in the previously referenced patent applications.

An analog film memory can be used as an image database memory. The analog film memory can provide large capacity storage, such as hundreds of millions of bytes of storage to billions of bytes of storage per memory. Such analog film memories are manufactured by Bendix and other companies. It can be implemented by recording the map images on film, mechanically scrolling the film, and scanning the film with a flying spot scanner to convert from optical to electrical image form.

A digital RAM can be used as an image database memory. The digital RAM can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Such RAMs were manufactured by Intel, Texas Instruments, Mitsubishi, and other companies. Digital RAMs are described in the previously referenced patent applications.

A digital ROM can be used as an image database memory. The digital ROM can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Such ROMs are manufactured by Intel, Texas Instruments, and other companies. Digital ROMs are described in the previously referenced patent applications.

A digital PROM can be used as an image database memory. The digital PROM can provide large capacity storage, such as hundreds of millions of bytes of storage per memory. Such PROMs are manufactured by Intel, Texas Instruments, and other companies. As discussed for RAMs, ROMs, CCDs, and other solid memories; particular advantages are achieved with PROMs. PROMs have the advantages of integrated circuit ROMs, such as being solid state and non-volatile and PROMs have the advantages of integrated

circuit RAMs, such as being electrically alterable.

Various database memory arrangements have been individually described. Such database memory arrangements can be used in combination to provide additional advantages. For example, an integrated circuit ROM can provide a low cost, non-volatile, monolithic database memory. However, it does not provide the capability of convenient loading of updated information. Combining of an integrated circuit ROM, such as for storage of permanent terrain information, in combination with an integrated circuit alterable memory, such as for storage of threat information, provides the advantages of a ROM for permanent images and an alterable memory for changing images. This permits loading of threat information prior to a mission with the alterable memory, overcoming an electrical alterability limitation of the read-only memory.

Many applications need ruggedized systems; such as military applications and avionic applications. The moving map display application discussed herein is representative. The image processor arrangement discussed herein is implemented with electronic circuits and hence is easily ruggedized. The database memory may not be implemented with integrated circuits and hence may be more difficult to ruggedize.

Rotating memories, such as disk and drum memories, have delicate mechanical assemblies and therefore are sensitive to shock, vibration, and other environmental considerations. However, rotating memories have been implemented in rugged form, such as for avionic applications. Also, rotating memories, such

as Winchester disk memories and optical disk memories, which are readily available in non-rugged configurations, are presently being ruggedized for military and avionic applications.

Magnetic tape memories, as with rotating memories, have delicate mechanical assemblies and hence are sensitive to shock, vibration, and other environmental considerations. However, tape memories have been implemented in rugged form, such as for avionic applications. Also, tape memories, such as cassette tape memories, which are readily available in non-rugged configurations, are presently being ruggedized for military and avionic applications.

Solid state memories; such as RAM, ROM, PROM, CCD, and bubble memories; provide ruggedness comparable to the electronic circuit portions of the system. Solid state memories often have lower capacity and greater cost per bit compared to rotating memories and tape memories. Improved capacity and reduced cost is obtained with the analog CCD memories and the analog bubble memories described in the referenced patent applications. Therefore, from a cost and capacity standpoint, the analog solid state memories are the more advantageous than the digital solid state memories.

*f* Electrical alterability is a desirable feature for database memories, such as to permit electrical modification of the images. Database memories that are not electrical alterable may involve mechanical operations to change the database; such as removing and replacing a read only optical disk or a solid state ROM module. Of the above described memories; electrically alterable memories include magnetic disks, magnetic drums,

magnetic tape, RAMs, and alterable CCDs. Of the above described memories; non-electrically alterable memories include some types of optical disks and ROMs.

Non-volatility is a desirable feature for database memories, such as for preserving stored information during power turn-off. Database memories that are volatile may include battery back-up to preserve stored information during power turn-off, or may require non-interruptable power sources, or may require reloading of the memory. Of the above described memories; non-volatile memories include magnetic disks, magnetic drums, magnetic tape, ROMs and non-alterable CCDs. Of the above described memories; volatile memories include RAMs and alterable CCDs.

### Digital Video Disk Recording

Video disks are typically analog storage devices for storage of analog video information. Storage of digital information on a video disk can provide important advantages. For example, the image processing system discussed herein may use digital processing in conjunction with a digital memory map stored on a digital video disk to provide dynamic effects. Storage of analog video information on a video disk may include an analog to digital converter (ADC) to convert from the analog information on the video disk to digital information for storage in a digital memory map. Alternately, storage of image information in digital form on a video disk can simplify interfacing between a digital processor and a video disk, such as overcoming the need for an ADC.

Digital information can be stored directly on an analog video disk. For example, information stored on a video disk can be either in the 1-state or the 0-state as digital information rather than in a continuously variable range of analog information.

Video disks are primarily sequential recording media. However, many video disks have a form of random access capability, permitting the read head to be positioned across tracks. Head positioning may not be as precise as desired for digital recording, such as positioning to a single selected track. Further, the bit error rate (BER) may not be as good as desired for digital recording. A configuration will now be discussed to overcome such limitations of video disks associated with digital recordings.

A record of information on video disk can be identified with a digital code, such as a preamble or header at the beginning of the record, and can be terminated with a postamble having a digital code. For example, the header may have a Barker code or other uniquely identifiable code to identify the beginning of a record and can have a record identification code, such as following the Barker code, for identifying the record being read. Records can be structured as one record per track for convenience of discussion. Alternately, records can be structured as a record per fractional track, a record per multiple tracks, or a record per multiple and fractional tracks.

Head positioning inaccuracies can be overcome by positioning of the head to the approximate position of the desired track, within the accuracy of the head positioning mechanism. The identification code of the track can be read and a determination can be made if the track that has been located (a) is the desired track, (b) is prior to the desired track, or (c) is subsequent to the desired track. If the desired track has been located, it can be read as needed. If a prior track has been located, the video disk can be read until it has advanced to the desired track. If a subsequent track has been located, the head can be moved backwards to the desired track. Alternately, if a subsequent track has been located, the head can be moved backwards to a prior track preceding the desired track and then can be advanced to the desired track as discussed above.

Read errors can be overcome by reading and re-reading the same information and by comparing the multiple reads of the same information to detect and correct errors. For example; video disk systems often provide a single frame capability, where a single video track can be read and re-read repeatedly to display the same frame on a video monitor. Similarly, in a digital video disk configuration, the same digital information can be read and re-read for error correcting purposes.

### Mosaic Storage Locations

The particular features of the present image processor invention facilitate use of a video tape recorder for a database; such as for dynamic interactive scenarios in arcade games and in training simulators. For example, the present invention can be implemented to load mosaics (or frames) relatively infrequently from the database memory with a virtual scrolling architecture. Also, the present invention can generate dynamically modified mosaics to the display at a real-time frame rate by image processing of static mosaics from the image memory to generate multitudes of dynamic frames in real-time to the display. A video tape recorder can be used for a database for the present image processor even though the video tape access has sequential access because many applications need relatively low traffic between the database memory and the image processor.

Optimum placement of mosaics in the database memory reduces time to access additional mosaics and facilitates use of low access rate database memories, such as a video tape having sequential access. Placement of mosaics can be adapted to reduce search time on the database memory for accessing of additional mosaics, such as to facilitate branching and virtual scrolling. For example, mosaics can be accessed from video tape for overlaying into a wrap-around architected image memory. The image memory can be a 2-dimensional memory map and the video tape can be a 1-dimensional sequence of mosaics. Moving over the memory map in any direction in the 2-dimensional plane of the memory map can require accessing of mosaics from the video memory in the direction of motion for overlaying with the virtual

scrolling implementation. Because motion over the memory map is preferably in any direction, such as under operator control; each mosaic permits motion into 8-adjacent mosaics. Therefore, it is desirable to have easy access to all of the 8-adjacent mosaics stored on video tape from the mosaic in image memory. Similarly, when one of 8-adjacent mosaics are accessed from video memory and overlayed into image memory, it is desirable to have easy access to all of the 8-mosaics adjacent thereto stored in video memory from this new mosaic in image memory. For random access-type-database memories, such as disks; enhanced sequential access is desirable but has reduced importance. For sequential access-type database memories, such as video tape; enhanced sequential access is desirable and has increased importance. Enhanced sequential access can be provided by grouping of mosaics in a manner consistent with the motion through the environment in accordance with the virtual scrolling implementation.

It becomes apparent that the number of mosaics stored along a sequential access database memory increases exponentially as a function of the distance away from the start pixel. This is a function of the square-law type effect of the number of scenarios, where the motion from a start mosaic can go to each of 8-adjacent mosaics, and then to each of 8-adjacent mosaics for each of those, and so forth. Hence, the tape very rapidly fills up close to the start mosaic. Therefore, it may be desirable to define directions and to move away from the start mosaic. For example, the start mosaic can have the adjacent mosaics, possibly 4-mosaics deep, the mosaic placed adjacent thereto, as motion

proceeds further in depth into 4-quadrants. A high speed tape wind operation can be performed to go to a rather remote location on the tape to pick up subsequent mosaics in that direction.

The tape is effectively 1-dimensional. The additional directions of a 2-dimensional memory map can be provided by having relatively widely spaced blocks of mosaics, effectively dividing the tape into a multi-dimensional format.

One feature is to place redundant mosaics on the tape to reduce search efforts. For example, if the plus-X and plus-Y direction (first quadrant) mosaics are grouped together at <sup>A</sup>~~X~~ separate locations on the tape, there are mosaics along the 45-degree line and adjacent thereto separating these 2-directions that should be provided for both directions of motion. Rather than have the tape search back and forth between 2-blocks of mosaics as motion traverses along this diagonal, overlapping mosaics can be replicated around the boundaries of the different directions to minimize such searching.

As discussed above, accessing of mosaics from a database in response to geometric processing can be enhanced by optimum storage of the mosaics in the database. A mosaic can be configured as an array of pixels, such as a 2-dimensional array of 512-pixels by 512-pixels for a mosaic. Mosaics can be accessed from the database, preprocessed, and loaded into image memory in response to geometric processing; such as in response to translation that translates towards the edge of the mosaics in image memory and such as in response to compression that compresses the mosaics in image memory.

A large database image can be partitioned into a 2-dimensional array of mosaics and each mosaic can be partitioned into an array of pixels. For example, a database image can be partitioned into an array of 8-mosaics by 8-mosaics (Fig 7A) and each mosaic can be partitioned into an array of 512-pixels by 512-pixels. Accessing of mosaics from database memory will generally be accessing of adjacent mosaics. For example, as shown in Fig 7A; mosaics 27, 28, 35, and 36 can be stored in image memory; where translation to the right can involve accessing of mosaics 29 and 37 and translation upward can involve accessing of mosaics 19 and 20. Consequently, accessing of adjacent mosaics can involve accessing in a 2-dimensional environment.

Database memories may be implemented in 1-dimensional form. For example, a digital magnetic tape memory may store pixels sequentially to store a mosaic and may store mosaics sequentially to store a database image. A video tape memory may store mosaics sequentially to store a database image. A disk memory, such as a Winchester disk memory or a video disk memory, may store mosaics sequentially on a track and may sequentially access different tracks. Therefore, an arrangement of storing a 2-dimensional array of mosaics with a sequential or a 1-dimensional database can be advantageous.

An arrangement for storing a 2-dimensional array of mosaics in a 1-dimensional or sequential form is shown in Figs 7A and 7B. The database memory can be organized to store 1-dimensional groupings of mosaics in one direction in adjacent blocks of memory and to store 1-dimensional mosaics in the other direction in adjacent mosaic locations. For example, horizontally adjacent

mosaics 1 to 8 in the 2-dimensional array (Fig 7A) can be stored as adjacent mosaics in the 1-dimensional memory (Fig 7B).

Vertically adjacent groupings of mosaics; i.e., mosaic grouping 1 to 8 and mosaic grouping 9 to 16; can be stored as adjacent groupings of adjacent mosaics; i.e., sequential mosaics 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, and 16 (Fig 7B). In this configuration, horizontally adjacent image mosaics are also database adjacent mosaics and vertically adjacent image mosaics are database non-adjacent mosaics stored a number of mosaics storage locations away from adjacency.

For example, in the arrangement shown in Figs 7A and 7B having a square 8-mosaics by 8-mosaic image stored in a 1-dimensional database memory; horizontally adjacent image mosaics are stored in database memory in adjacent locations and vertically adjacent image mosaics are stored in database memory 8-mosaic storage locations away. Consequently, such a square image arrangement can be stored in database memory with a maximum search distance for any adjacent mosaic being a square root type function of the total number of mosaics in the image.

For example, in the arrangement shown in Figs 7A and 7B, the maximum search distance for vertically adjacent mosaics is 8-mosaics; which is the square root of the quantity of 64-mosaics in the image. Also, for an example of 1-billion pixels in database memory partitioned as 3600-mosaics each having 512-pixels by 512-pixels; the 3600-mosaics can be partitioned in an array of 60-mosaics by 60-mosaics and consequently will have a maximum search distance of 60-mosaics (the square root of 3600).

Similarly, aspect ratios other than square aspect ratios can be configured; where for example the above 3600-mosaic arrangement can be configured as 100-mosaics for horizontal adjacency and 36-mosaics for vertical adjacency with vertically adjacent mosaics being stored in adjacent locations in the 1-dimensional database memory and with horizontally adjacent mosaics being stored 36-mosaic locations apart.

Accessing of a mosaic can be achieved by identifying the address of the storage location for the mosaic. For simplicity of discussion, the addresses of the mosaic in a 1-dimensional database memory will be numbered sequentially (Fig 7B). The 1-dimensional mosaic addresses can be mapped into the 2-dimensional image of mosaics having sequentially addressed mosaics in the horizontal direction and having non-sequentially addressed mosaics in the vertical direction (Fig 7A). Mosaic addresses can be derived in various forms. For example, addresses of adjacent mosaics can be calculated or can be stored. Addresses of adjacent mosaics can be stored with the current mosaic; where a current mosaic, mosaic-28, can store the address for horizontal right translation (mosaic-29), the address for horizontal left translation (mosaic-27), the address for vertical up translation (mosaic-20), and the address for vertical down translation (mosaic-38). Also, addresses of adjacent mosaics can be calculated; where horizontal right translation from current mosaic-28 can be calculated by incrementing the mosaic address from current mosaic-28 to mosaic-29; where horizontal left translation from current mosaic-28 can be calculated by decrementing the mosaic address from current mosaic-28 to mosaic-

27, where vertical up translation from current mosaic-28 can be calculated by subtracting 8-addresses from current mosaic-28 to mosaic-20, and where vertical down translation from current mosaic-28 can be calculated by adding 8-addresses from mosaic-28 to mosaic-36. Other arrangements for storing adjacent mosaic addresses, for calculating adjacent mosaic addresses, and for otherwise deriving adjacent mosaic addresses can be implemented.

### Mosaic Shape

Various arrangements are described herein for overlaying image information from a database memory into a memory map. For example, mosaics accessed from a video disk can be overlaid on portions of image memory to implement wrap-around and virtual scrolling. Various alternate methods of implementing overlays will now be discussed.

Video disk recordings conventionally record a frame having the near-square aspect ratio of a television receiver. Although any reasonable aspect ratio mosaic can be used in accordance with the present invention; including near-square aspect ratio mosaics, image memory aspect ratio mosaics, or window aspect ratio mosaics; it may be desirable to have highly non-square aspect ratio mosaics for certain configurations. For example, a configuration is shown in Fig 7C implementing mosaic frames having non-square aspect ratios. Image memory 752 and window 756 are shown having near-square aspect ratios for convenience of discussion. Window 756 is shown smaller than image memory 752 to permit translation and rotation of window 756 over image memory 752. Window 756 represents the portion of the image in image memory 752 that will be scanned out, such as to an image memory map or to a raster scan monitor. As discussed above, window 756 can be rotated and translated over image memory 752 to facilitate such scan-out. As window 756 translates toward an outer boundary of image memory 752, mosaic frames are accessed from the database and inserted into image memory 752 to extend the outer boundary being approached by window 756.

Mosaic elements may have an aspect ratio different from the aspect ratio of image memory 752 or window 756. As shown in Fig 7C, mosaic elements 750 overlayed on the right and left hand sides of image memory 752 may be tall and narrow to permit greater coverage in the Y-direction than in the X-direction to minimize the number of mosaics accessed from the database to extend image memory 752 an increment in the X-direction. For example, if mosaics are equal in height to the height of image memory 752, then only a single X-direction mosaic need be accessed from the database to extend image memory 752 an increment in the X-direction. Similarly, if mosaics are equal to one half of the height of image memory 752, then two X-direction mosaics need be accessed from the database to extend image memory 752 an increment in the X-direction. Similarly, as shown in Fig 7C, mosaic elements 754 overlayed on the top and bottom of image memory 752 may be short and wide to permit greater coverage in the X-direction than in the Y-direction to minimize the number of mosaics accessed from the database to extend image memory 752 an increment in the Y-direction. For example, if mosaics are equal in width to the width of image memory 752, then only a single Y-direction mosaic need be accessed from the database to extend image memory 752 an increment in the Y-direction. Similarly, if mosaics are equal to one half of the width of image memory 752, then two Y-direction mosaics need be accessed from database to extend image memory 752 an increment in the Y-direction.

As discussed with reference Fig 7C, it may be desirable to access a group of pixels having an aspect ratio that is tall and narrow or short and wide. This can be implemented by accessing a column of pixels for a tall and very narrow aspect ratio mosaic and by accessing a row of pixels for a wide and very short aspect ratio mosaic. As discussed above, linearizing a memory map in the database can be implemented with concatenated rows or concatenated columns. For concatenated rows, the pixels in a row and the sequence of rows are organized for convenient accessing of wide and very short aspect ratio mosaics but are organized in an inconvenient manner for accessing of tall and very narrow aspect ratio mosaics. Conversely, for concatenated columns; the pixels in a column and the sequence of columns are organized for convenient accessing of tall and very narrow aspect ratio mosaics but are organized in an inconvenient manner for accessing of wide and very short aspect ratio mosaics. Therefore, each concatenated organization facilitates one aspect ratio mosaic and does not facilitate the other aspect ratio mosaic.

In order to provide for efficient overlay mosaic generation, a database can be formed having duplicate versions of the same image, one concatenated in the column direction and the other concatenated in the row direction. In this manner, overlay mosaics can be efficiently accessed that are both, tall and narrow and short and wide, by accessing the overlay mosaics from the appropriate one of the two stored images.

### Continuous Database Images

An arrangement has been discussed for storing an image in a database in the form of mosaics. Alternatively, an image can be stored in the database as a single continuous image, without being partitioned into a plurality of mosaics. For example, an image can be formed with a group of pixel words, each pixel word identified by the X-coordinate and Y-coordinate related to the location of the pixel word as it appears in the 2-dimensional image. The supervisory processor can keep track of the position of the image in the memory map, such as by keeping track of the coordinates of the 4-corners of the image in the memory map, and the supervisory processor can access lines of pixels or columns of pixels from the database to overlay in the memory map for virtual scrolling. This may be considered to be a mosaic arrangement; where a mosaic is reduced to a pixel, or a column of pixels, or a row of pixels, or other grouping of pixels. The image can be stored in the database in memory map form in a fashion similar to that discussed for memory map storage in the image memory map, except that the image in the database may comprise a vary large memory map compared to image memory.

A computer memory, such as a RAM memory and a disc memory, are typically single dimensional memories having continuous addresses. An image is typically a 2-dimensional array of pixels having row addresses and column addresses. A single dimensional memory can be used to implement a 2-dimension memory map by linearizing the memory map for storage in the single dimensional (linear) computer memory. A memory map can be linearized by concatenating the rows to form a single-dimensional long row or

by concatenating the columns to form a single dimensional long column.

## Database Information

The database memory can store various types of information. Intensity information, either monochromatic or color intensity information, can be stored in the database memory. Intensity information can be displayed directly <sup>or</sup> ~~of~~ can be modified; such as with a lookup table, spatial filtering, or other processing. Altitude information, such as altitude of each pixel, can be stored in the database memory, such as for topological occulting and shadowing. Occulting information, such as altitude information, can be stored in the database memory. Control information; such as occulting flags, visibility flags, edge flags, and surface ID information; can be stored in database memory.

Database information can be stored in compressed or in non-compressed form. Processing of non-compressed information has been previously described. Processing of compressed information can include de-compression of the compressed information and then processing of the de-compressed information as non-compressed information. Various data compression and de-compression techniques are well known in the art.

Processing of database information can be performed in various ways; such as discussed herein for processing of intensity, altitude, range, and control information.

### Database Data Compression

The amount of storage in the database for a particular image can be reduced by storing compressed image data. The system can be implemented to decompress the image for image processing and display.

Images are typically processed as an array of pixels, where each pixel stores information associated with a point on the image that can be displayed. There is often continuity between pixels. For example; intensities, altitudes, and other parameters that can be stored in a pixel often have gradual variations in magnitude from pixel to pixel. The data can be compressed by storing parameters, such as coefficients, related to the gradual variations between a group of pixels. Defining of the variations between pixels permits information pertaining to the variations for a smaller number of compressed points to be stored in place of the information pertaining to a larger number of pixels, yielding compressed information. Information can be decompressed by regenerating the pixel parameters from the compressed information; such as by extrapolating or interpolating the compressed information to obtain decompressed information. One form of compression stores variations between pixel parameters and one form of decompression reconstructs pixel parameters from the variation information.

Many data compression techniques can be used. For example, image points can be stored in the form of coefficients of an interpolation or extrapolation equation to provide interpolation between datapoints or extrapolation from a datapoint. These coefficients can be higher order coefficients; such as second

order, third order, and still higher order coefficients. The coefficients can be 1-dimensional, 2-dimensional, 3-dimensional, and higher dimensional coefficients.

In a 1-dimensional configuration, the data~~s~~ can be interpolated between points or extrapolated from points in a single dimension; such as in a radial direction or in a rectilinear dimension. Each of the coordinates; such as the 2-coordinates in the plane (X and Y) of the image and the magnitude coordinate (Z); can each be interpolated or extrapolated separate of the other coordinates. Alternately, the coordinates can be interpolated or extrapolated in combinations of different dimensional combinations, such as a 1-dimensional interpolation or extrapolation for altitude or Z and a 2-dimensional interpolation or extrapolation for X and Y, or latitude and longitude, or R and theta.

In a 2-dimensional configuration; the image can be interpolated between points or extrapolated from points in a 2-dimensional arrangement; such as in a 2-dimensional rectilinear X-direction and Y-direction together with a 1-dimensional rectilinear magnitude direction. Each of the coordinates; such as the 2-coordinates in the plane of the image (X and Y) and the magnitude coordinate (Z); can each be interpolated or extrapolated in pairs of coordinates. For example, the X, Y, and Z-coordinates can be processed as 2-dimensional XY, YZ, and ZX 2-dimensional interpolations or extrapolations. Alternately, the coordinates can be interpolated or extrapolated in combinations of different dimensional combinations, such as a 1-dimensional

interpolation or extrapolation for altitude or Z and a 2-dimensional interpolation or extrapolation for X and Y, or latitude and longitude, or R and theta.

In a 3-dimensional configuration, the data can be interpolated between points or extrapolated from points in a 3-dimensional arrangement; such as in a 2-dimensional rectilinear X-direction and Y-direction together with a magnitude direction. Each of the coordinates; such as the 2-coordinates in the plane of the image (X and Y) and the magnitude coordinate (Z); can all be interpolated or extrapolated in a 3-dimensional configuration. For example, the X, Y, and Z coordinates can be processed as a 3-dimensional XYZ interpolation or extrapolation.

Compressed information can be decompressed prior to loading into image memory so that images in image memory are in decompressed form. Alternately, compressed information can be stored in image memory and can be decompressed when processed as an output of image memory. Geometric processing can be performed on decompressed information stored in image memory; where decompression can be performed prior to geometric processing, such as before loading into image memory or after accessing from image memory. Geometric processing can be performed on compressed information in image memory; where decompression can be performed subsequent to geometric processing, such as before spatial processing or after spatial processing.

Decompression can be implemented with incremental processors. For example, an incremental processor can be implemented to provide 1-dimensional, 2-dimensional, 3-

dimensional, or higher dimensional interpolation or extrapolation using first order, second order, third order, and higher order processing. For example, cubic interpolation between points can provide multitudes of intermediate points having a third order interpolative relationship therebetween.

DISPLAY APPLICATIONS

### General

Display applications for the image processing disclosed herein are many and varied. Most display applications currently satisfied with CG and CIG systems can use the present image processing system of the present invention, either in replacement of or in addition to the CG and CIG capability. These CG and CIG applications include CAD/CAM, simulation, industrial control, graphic work stations, graphic arts, computer aided instruction (CAI), and many others.

Most display applications needing image processing capability can use the present image processing system of the present invention. These image processing applications include medical image processing, video special effects, and many others.

A brief discussion of typical applications will now be provided, followed by more detailed discussions of representative applications. The system of the present invention can be used for training of personnel, such as a simulator; as a display for a vehicle, such as for an aircraft cockpit display; for investigation and evaluation of large dynamic range database information, such as with LANDSAT images; and for many other applications. A training simulator application is characterized by a drone aircraft having a remotely controlled video camera for communicating with a ground-based crew. Camera images can be simulated; dynamically driven as a function of vehicle roll, pitch, and yaw dynamics; vehicle altitude; and other parameters. A map display can be provided for displaying a terrain map, such as on a cockpit display for a pilot. The map can be dynamically updated as a function of vehicle latitude and longitude and the

map display can also be dynamically modified to simulate heading, roll, pitch, and other dynamic parameters. This permits a pilot to preserve his perspective of the outside world by viewing a terrain image that changes in spatial <sup>compression</sup> ~~compression~~ as a function of altitude, changes in orientation as a function of heading, and that provides tilting capability with range-related spatial compression as a function of pitch and roll. Such a map display can have superimposed overlays, such as topographical overlays for terrain and radar overlays for registering of radar images with terrain map images.

Image evaluation applications can store a large image having high dynamic range in database memory, can selectively pan through the image to select the appropriate area, and can zoom on the image to select the appropriate magnification to investigate details and to preserve the observer's perspective. For example, panning over a highly compressed image preserves the perspective from which the observer is viewing and then zooming on a point of interest provides high detail for a particular point within that image. Then zooming back out and again panning permits investigation of other areas of interest.

In a medical application, the medical investigator can investigate tomographic, X-ray, and ultrasound images. Investigation can be provided by accessing image information, such as mosaics pertaining to a large image, from database memory and providing rotation, translation, zoom, and other image processing capabilities to enhance diagnosis. For example, a large highly detailed medical image, such as a tomographic image,

can be stored as a group of mosaics in database memory. A medical investigator can control the system to roam through the image and zoom on key points of the image, similar to the image evaluation application discussed above. The medical investigator can zoom out to see a large portion of the image to provide a perspective, roam across the image to an area of interest, and then zoom in on the area of interest for more detailed investigations. The zooming out can be performed with spatial compression, as discussed for other applications herein.

In an animation application; images, such as background images, can be stored in database memory and accessed as the animation scenario progresses. Images can be overlaid with graphics; such as cartoon characters, vehicles, trees, and other graphics. As the scenario progresses, the detailed images accessed from video disk can be translated, rotated, spatially compressed or decompressed, and otherwise processed to enhance the animation scenario. For example, graphic overlays can be provided for translating across the background, being overlayed on background images which are rotated and translated in conjunction with the viewport position and spatially compressed and decompressed as a function of range and other parameters. Also, synthesized terrain images can be processed with translation, rotation, and spatial compression or decompression to animate motion across terrain from an airplane, similar to that discussed for the simulation application herein. Processed images in animation can include terrain, clouds, water, and other texture type images. This animation feature can be used for video games, movies, television, and other applications.

In a satellite application, an investigator can investigate land resources, military groupings, weather, and other such images using methods discussed above for image evaluation applications.

### Moving Map Display Application

A map display is a particular application of a large database system. Large database systems are described herein; such as for remote sensing, cartography, and printing applications.

Map displays can be moving map displays, such as in a military aircraft, or static map displays, such as in a remote sensing or cartographic application. Moving map displays can be used in various platforms including spacecraft; aircraft, such as in fixed wing fighter aircraft and in helicopters; ground vehicles, such as tanks and trucks; seacraft, such as surface ships and submarines; stationary locations, such as buildings and field stations; and in other platforms. Static map displays can be used in various platforms including buildings and field stations and in moving platforms; such as in spacecraft, aircraft, ground vehicles, and seacraft. The moving map display for aircraft discussed in detail herein is representative of the other map display implementations.

A moving map display system can be implemented with the image processor discussed herein. Mosaic map images can be stored in a database, such as a video disk or video tape database, and can be fetched and displayed in accordance with the virtual scrolling configuration discussed herein. Observer motion, such as motion of an aircraft, is simulated with a map display. For example, as the aircraft flies in a particular direction, translation of the image processor window in the related direction provides translation of the map in the display as the aircraft flies over the related terrain. As the aircraft

changes heading, rotation of the image processor window provides rotation of the map in the display as the aircraft rotates over the related terrain. As the aircraft increases altitude, compression of the image processor window provides compression of the map in the display as the aircraft flies over the related terrain. As the aircraft decreases altitude, expansion of the image processor window provides expansion of the map in the display as the aircraft flies over the related terrain. As the aircraft banks, range variable compression of the image processor window provides 3D-perspective of the map in the display as the aircraft flies over the related terrain.

F In some configurations, it may be desirable to reduce the information in the database. Such configurations can be implemented with lower detailed images, such as graphic images, supplemented with higher detailed images, such as image processing video images, for areas of higher interest, such as airport areas. Lower resolution areas can be implemented with a graphic database portraying a lower detail map image and higher detail images can be implemented with a video database with image processed high detail textured terrain images.

The higher detail textured images stored in database can be generated by digitizing existing images. Aerial photographs provide an excellent source of images for digitizing. Also, government agencies, such as the Defense Mapping Agency (DMA), have extensive maps and images of the world.

The moving map display can be implemented with geometric manipulation of map images from a database as if the display is a special optical window in the floor of the cockpit. As the aircraft maneuvers, the image processor processes static map images accessed from the database to generate dynamic images consistent with aircraft maneuvers. For example, as the aircraft translates, the map image is scrolled across the viewport; as the aircraft changes heading, the map image is rotated in the viewport; and as an aircraft increases and decreases altitude, the map image is compressed and expanded, respectively. Therefore, the map display acts as if it were a near-photographic quality "window" in the bottom of the aircraft.

The moving map display is an all-weather "window" that provides ground images independent of real world visibility; is a digitally controlled window that permits simulated flight training and mission evaluation as time is available during flight; and is a flexible window permitting overlays, symbols, and other navigation aids to be overlayed.

The map display can be provided in high detail form to pixel resolution, limited only by the resolution of the CRT monitor and the digitized imagery in the database.

Special visual effects can be provided for improved operator performance. For example, the geometric operations of rotation, translation, expansion, and compression are supplemented with a 3D-perspective capability. 3D-perspective provides range-related compression consistent with 3D-perspective. Therefore, as the aircraft banks, the image also "banks"; showing terrain off to the side of the aircraft towards the horizon. This side-looking

image is compressed as a function of the range toward a vanishing point that is beyond the horizon. Such visual cues aid in maintaining a pilot's perspective, such as during low visibility flight, and aid in visual operations, such as dead reckoning navigation and attack.

F Utilization of database memory can be enhanced by storing compressed information and then decompressing the informations, such as with an incremental processor, to obtain decompressed image information for image processing.

Symbolic images can be overlayed on the pictorial image. Symbols can include navigational symbols; such as topographical lines, checkpoint identification, and alphanumeric legends. Symbols can also include "pathway in the sky" images; such as flight corridor images.

Overlays provide for improved operator interfacing. Symbols can be overlayed for characterizing pictorial features; such as cultural features and military threats. Alphanumerics can be overlayed for characterizing pictorial features and providing supplementary information, such as describing selected features. Graphical images can be overlayed; such as a pathway in the sky type of overlay to guide a pilot through an environment.

Overlays can be provided as a function of altitude; such as an altitude parameter stored in the pixel word. These overlays can be stored in the database or can be generated from altitude information. For example, topological constant altitude lines can be stored in the database, either overlayed on the images in the

database or non-overlaid in the database for overlaying in the image processor. Alternately, topological constant altitude lines can be generated in the image processor in response to altitude information stored with the image. For example, the image processor can scan the image, process altitude parameters, and generate constant altitude lines similar to those seen on a topological chart. Such constant altitude lines can be generated in an interface buffer memory prior to loading into image memory, or during loading into image memory, or after loading into image memory. Similarly, other altitude-related information can be stored as overlays in database memory or generated by the image processor for overlaying on the image. Such other altitude related information can include cultural information and can include altitude-related colors, similar to the altitude related colors on a topological chart.

Overlays into image memory was previously discussed relative to scrolling toward an edge of image memory. Other conditions can cause the displayed to move toward and through an edge of image memory, such as rotation and compression. For example, rotation of a window that is near an edge and that is aligned with an edge of image memory can cause a corner of the window to rotate through an edge of image memory. Also, compression effectively expands the window, such as with undersampling; where an expanding window can cross an image memory boundary. Hence, virtual scrolling from database memory can also be controlled as a function of rotation and compression; similar to that previously described for translation.

Altitude information can be stored with an image in compressed or in non-compressed form in database memory. The altitude information can accompany intensity information loaded into an interface buffer memory and can be used for performing altitude-related pre-processing, such as shadow processing and topological line and color overlays, in the buffer memory. The altitude pre-processed image can be loaded into image memory for subsequent geometric processing. If subsequent altitude processing is to be performed, such as topological occulting processing; then the altitude information can be loaded into image memory. However, if subsequent altitude processing is not to be performed; the altitude information need not be loaded into image memory; thereby reducing the word size in image memory.

### Informational Database Application

A database can be implemented having information that is first accessed and then image processed for presentation or for recording. The database can include learning materials, such as for an educational application; an encyclopedia or other reference materials, such as for a research application; service manuals, such as for an automobile repair application; or other materials. Material can be selected by an operator, automatically accessed from the database by the supervisory processor, and image processed for recording or for presentation by the dynamic image processor. For example, a student can obtain self-learning instructions by selecting a physics teaching aid stored in the database and controlling processed images thereof. Geometric processing can provide motion effects to enhance teaching to facilitate improved understanding.

Computer databases containing information in digital form are used extensively in the computer industry. For example, video disks can be used for storing such information. Video disks can make feasible the storage of images together with the storage of text information. As discussed herein, the database may include digital, analog, or combinations of digital and analog information. The visual systems discussed herein can access digital and analog information from a database for display. The images can be image processed to provide dynamically moving images from static database images.

Investigation of pictorial information can be enhanced with geometric processing. For example, an auto mechanic can select a diagram of an automobile for presentation and can then control

the diagram to translate, rotate, and change in size for investigation of parts of that diagram, such as for viewing from an improved perspective. Similarly, a medical diagnostician can select an X-ray image for presentation and can control the X-ray image to translate, rotate, and change in size for investigation of a part of the X-ray.

Educational pictorial information can be enhanced with geometric processing. For example, a student can select a diagram of an object and cause the object to move in accordance with physical principles to facilitate learning of those physical principles.

An encyclopedia application is representative of many other database informational applications; such as a dictionary applications, a financial application a technical report application, and other applications. An encyclopedia can be stored in a database, including text and static images. The image processing system can be used to process the images to provide dynamically moving images in response to the stored static database images. For example, a person accessing encyclopedia information can also access a related image and can dynamically manipulate this image in translation, rotation, expansion, and compression to better view the image and to provide additional <sup>esthetic</sup> and learning capabilities. A motion scenario can be stored in the database for automatically providing motion for the image. Therefore, static database information can be automatically converted to dynamic information for better presentation.

A technical report, proposal, or other technical document application is further representative of informational database applications. For example, a technical report can be stored in the database together with static images to document the report. A reviewer of the report can obtain dynamic image information in addition to text information with the image processor dynamically processing the static database images to provide dynamic images. For example, a report on development of an aircraft can be documented with static images of the aircraft. A reviewer can view the static images after dynamic processing to obtain a moving image of the aircraft flying.

### High Definition Television Application

High definition television (HDTV) involves significant data compression. This can be achieved by reducing the amount of redundant information communicated. In a television picture, a portion of the image bandwidth has to do with either static information or with information that has been geometrically modified. Usually, only a portion of the image pixels have been created or erased, such as with occulting and clipping. Therefore, many of the image pixels that have not been created or erased can be modified with an image processor in order to reduce the need to communicate such modified information. The present image processing invention can perform such processing to facilitate HDTV.

### Special Effects Application

A low cost special effects system can be used in a TV studio application, a business graphics application, and other applications. For example, special effects can be used to generate slides for business presentations and to generate graphic art images for advertisements. Although the final product of a graphic art system may be static photographs, real time operation permits an operator to efficiently and rapidly configure images. Real time operation and interactivity enhances productivity.

### Remotely Piloted Vehicle Application

Remotely Piloted Vehicles (RPVs) provide an application for illustrating various other applications of the image processor of the present invention. An RPV can have different types of sensors and a datalink for communicating sensor information to a ground station. Image processors on-board the RPV and in the ground station can enhance operations. For example, a video camera in the RPV can provide video images, a radar system in the RPV can provide radar images, a side-looking radar (SLR) in the RPV can provide SLR images, a forward-looking infrared (FLIR) system in the RPV can provide FLIR images, and other sensors in the RPV can provide other images. These images can be processed or pre-processed in the RPV, can be communicated to the ground station, and can be processed or post-processed on the ground.

Pre-processing in the RPV can perform various functions. Pre-processing permits data compression to improve communication bandwidth requirements. Pre-processing permits silent operation, where pre-processed image information can be stored in the RPV for later transmission or for a "data dump" after landing. Pre-processing can take the form of digital filtering to reduce redundancies, pattern recognition to identify meaningful image features and to derive the characteristics and geographical location of the features of interest, and other such implementations.

Ground-based processing and post-processing can perform various functions. Ground processing provides for data de-compression to reconstruct images transmitted in compressed form. Ground processing permits image enhancement for evaluation by a

surveillance operator; such as providing the ability to investigate an image with rotation, translation, compression, expansion, and warping; and the ability to enhance an image; such as providing the ability to enhance edges, enhance important image features, and recognize important image features.

Registration of images and adapting images for evaluation is an important capability. Images can be registered, such as with a reference image, by translating the images into translational registration therebetween, by rotating the images into rotational orientation therebetween, by expanding or compressing the images into scale registration therebetween, and by warping the images to facilitate registration between important features of the images. After registration has been accomplished; pattern recognition, such as automatic pattern recognition or operator visual pattern recognition, can be performed more effectively.

Warping of an image facilitates matching with references for pattern recognition. Image warping facilitates matching of reference images, such as topological maps and geographical satellite images. For example, airborne images that are distorted can be compensated; such as images that are distorted due to visual effects, i.e. range variable perspective; camera effects, such as lens distortions; geographical effects, such as earth curvature; and other such effects. Such image warping can correct distortions for improved operator interaction and for improved automatic pattern recognition.

Images can be recorded for processing and evaluation at a later time. Recording can be performed with conventional memory devices; such as RAMs, PROMs, magnetic disk memories, magnetic core memories, digital bubble memories, video tape, video disks, and other conventional memories; and can be performed with unconventional memory devices; such as the signature memory, analog CCD memory, and analog bubble memory disclosed in the CCD patent applications and patents by Gilbert P. Hyatt referenced herein. Recorded images can be played back and evaluated at a later time. For example, an operator at an image processing console can control processing of the recorded images to investigate the images; such as with rotation, translation, expansion, compression, warping, and spacial filtering.

In addition to implementing operational requirements, the system of the present invention can be used for implementing simulation requirements, such as for training simulators for RPV systems.

### Digital Video Camera Application

The image processor of the present invention can perform dynamic video camera-type operations on images that have already been captured and thus can be called a digital video camera. These dynamic operations include rotation, translation, expansion, and compression; such as performed with a mechanical video camera. Translation is similar to panning of a video camera, rotation is similar to rotating of a video camera, and expansion and compression are similar to zooming of a video camera. Therefore, an image can be captured, such as with a conventional mechanical video camera, in a basic image form; the image can then be panned, rotated, and zoomed digitally with the digital video camera as if the mechanical camera itself had performed these functions.

The digital video camera has important advantages over a mechanical video camera. A mechanical video camera can usually capture an image a single time because the image may no longer be available. Conversely, the digital video camera can iteratively modify a pre-captured image to provide multitudes of different forms of presentation without losing the opportunity for capturing the basic image itself. Also, a mechanical video camera merely records motion of an image and then is locked-in to the particular motion of that image in pre-recorded form. However, the digital video camera generates motion from a static image and provides multitudes of different scenarios of motion for this static image independent of the method of motion of the original object from which the image was derived. This provides various important capabilities not available with mechanical

video cameras. First, the digital video camera can generate a moving picture from a static picture. Second, the digital video camera can generate a controllable scenario that adapts a pre-recorded scenario to the particular needs of the user unconstrained by the pre-recorded scenario.

The digital video camera can be used in applications having a digitized image in 2-dimensional form. For example, it can operate on static video images, dynamic video images, synthesized graphic images, synthesized textured images, digitized static images, and digitized dynamic images. It is applicable to different types of systems; such as simulation, medical image processing, and video image processing.

A TV receiver and a video recorder can use the digital video camera in conjunction with television viewing to modify the received video images. For example, a viewer can zoom in on a particular portion of an image for more detailed viewing, zoom out for broader coverage, rotate the image for special effects, and translate the image to better center on the items of interest.

There are many sources of images for the digital video camera. Digital images can be entered in memory map pixel form, such as in a raster scan form, to be captured in image memory. Analog signals in raster scan form can be digitized and captured in image memory. Images in printed form or visual form can be digitized with well known digitizing equipment and stored in image memory, such as computer graphic images.

### Landscape Architecture Application

A landscape architecture application is illustrative of various applications of overlaying objects to synthesize an image. Landscape architecture can include placement of shrubs, trees, bushes, grass, fountains, walls, and other plants and structures in an environment to provide an <sup>e</sup>sthetically pleasing environment and to provide the desired atmosphere. Landscape architecture is often performed with drawings. Drawings are relatively inflexible and have relatively low detail.

An image processing system will now be discussed that facilitates flexibility and high detail in landscaping. A database of objects can be constructed; such as shrubs, trees, bushes, grass, fountains, walls, and other plants and structures. Database construction can take the form of photographing such objects, digitizing the photographs, and storage of the digitized photographs in a video database. The video database can be a video disk, video tape, digital disk, digital tape, or other storage media. Architect controls can be joystick, keyboard, trackball, and other controls for image manipulation.

An image can be constructed by selection of a background image and overlaying objects on the image interactively by the architect. For example, the architect can select a shrub as an overlay object. The image processor will access the selected image from the database and store it in an overlay image memory. The architect can rotate, translate, expand, compress, and color the overlay object for overlaying on the background image. Overlaying of many objects permits a complex landscape image to be constructed. The architect can interactively change the

overlay objects to improve the landscape image. For example, a shrub can be reduced in size and translated to a different position with an occulting priority set relative to other overlay objects.

Occulting priorities can be assigned to overlay objects as they are selected and accessed from video memory. For example, an occulting priority can be assigned with a keyboard entry. Also, occulting priorities can be assigned by slewing of range, such as with a joystick. Occulting priorities can be used in accordance with the overlay configuration discussed herein; such as for selecting which one of a plurality of overlay images that fill a particular pixel will be displayed.

Colors can be assigned to overlay objects and can be modified under architect control. For example, intensities can be selectively increased and decreased; such as increasing of the red intensity by a first factor and decreasing of the green intensity by a second factor.

### Video "Photograph" Application

The Videonic system permits video pictures to be recorded with an inexpensive camera, enhancement as if the pictures were recorded with an expensive camera, and interactive real time viewing with exotic special effects.

Video pictures can be recorded on video tape with relatively low precision equipment. The recorded pictures can be enhanced and then displayed to higher precision with increased spatial resolution, such as with a larger number of pixels; increased color resolution, such as with a larger pallet; improved sharpness, such as with edge enhancement; improved smoothness; and reduced noise.

The viewer can interactively control special effects; such as panning, zooming, and warping of the pictures. Interactive viewing is performed with the image processor of the present invention, such as having expansion for viewing magnified features, compression for viewing a large area, rotation and translation to change viewing conditions, and warping to provide special effects. For example, facial closeups can be obtained from expanded group pictures and panoramic landscapes can be expanded to view interesting features, with substantial image enhancement.

Real time interactive viewing converts viewing from a spectator operation to an interactive participating operation. Pictures can be examined with scrolling and zooming. Pictures can be warped to obtain new and novel effects. Processed pictures that have been enhanced and expanded or compressed, translated, rotated, and warped can be permanently recorded, such

as on video tape.

Static viewing can be implemented with the freeze frame capability of a video memory, supplemented by image enhancement and interactive operation. Freeze frame jitter, common with video tape systems, can be eliminated. Geometric processing can be used to investigate a static image, such as positioning and zooming to enlarge a small feature to full screen size.

### Electronic Puppeteer Application

Animation is generally implemented by drawing of individual frames having motion therebetween and then photographing the frames and playing them back at high speed to simulate motion. Because of the extremely long times needed for generation of each of the frames, it is necessary to carefully choreograph and plan and draw the motion scenario to provide good motion appearance when played back at high speed. This process is expensive and time consuming.

An arrangement will now be discussed to provide high resolution detail with true motion generated at high speed with a minimum of pre-planning and choreography. Animation can be recorded in real time using one "puppeteer" or a plurality of "puppeteers" to control images to go through the scenario. For example, 3-puppeteers can each operate a set of joysticks and related controls to control an image overlayed on a background image and on the other images. Each of the 3-puppeteers can control a different image; where a first puppeteer can control a hero image, a second puppeteer can control a heroine image, and a third puppeteer can control a villain image. The puppeteers can each control their related image to interact with the other images and to execute a script scenario.

Interaction between images under control of puppeteers can be relatively spontaneous with a minimum of pre-planning and choreography. For example, a battle between a hero and a villain can be implemented under control of the puppeteers with minimum script direction; such as to generate a battle where the hero and the villain fight, resulting in the hero being knocked down

several times and eventually resulting in the hero overcoming the villain. The puppeteers can implement the battle and can intuitively control their respective images to provide a type of battle.

Another configuration that can be implemented is with the puppeteers in a simulated battle between themselves and instrumented to control the images to follow the puppeteers' motions. For example the puppeteers can have the main body joints; such as elbow, shoulder, neck, torso, knee, and hip joints; instrumented for control of the corresponding joints on the images to follow the scenario being performed by the puppeteers. Real time operation of the system is consistent with performance of daily activities in real time by the puppeteers and hence can be easy to implement with this system.

The above arrangement is an improvement over non-real time animation techniques, which are performed so slowly that they require extensive conditioning of the animators to provide such very slow motion frame generation. For example, an animator may spend an hour generating a frame, in contrast to the above electronic puppeteer arrangement which can generate a frame every 33-milliseconds.

### Flight Simulator Application

A flight simulator, such as an aircraft navigational simulator or bombing simulator, can be effectively implemented with the Videonic system. Ground terrain is simulated with digitized photographs. Occulting objects; such as tanks and trucks on the ground, lower flying aircraft, and smoke plumes; are generated with overlays. The overlays are independently manipulated to provide relative motion and 3D-perspective between overlays for realism.

### Traffic Accident Simulator Application

A traffic accident simulator will now be discussed as illustrative general applicability to other simulators; such as of a traffic simulator, an accident simulator, and other types of simulators. It may be desirable to simulate a traffic accident on an image processing system for the purposes of evaluating causes, for evaluating liability, for presentation to a court and jury, and for various other reasons. Such a simulator can be implemented with the image processing system of the present invention.

The background can be generated in various ways. In one way, an aerial photograph of the scene of the accident can be provided to simulate the actual environment. The photograph can be digitized and loaded into the database. This background image can be investigated with rotation, expansion, compression, and translation to provide insight into the accident environment; such as under operator interactive control and under computer control. The vehicles can be generated in various forms. In one form, each of a plurality of vehicles can be provided as individual overlays to simulate the actual vehicular environment. The vehicle overlays can be rotated, translated, expanded, and compressed to simulate accident dynamics; such as under operator interactive control and under computer control. In one configuration, overlay dynamics relative to the background image are under computer program control to follow a pre-defined accident scenario. The operator can then rotate, translate, expand and compress the background image and overlays, such as under joystick control, to permit investigation of the accident.

environment while the accident scenario is progressing. For example, a first car may be progressing in one lane of a road; a second car may be progressing in another lane of that road approaching the first car; other traffic, both vehicular and pedestrian traffic, can be progressing in accordance with the accident scenario. The scene can progress as the second car approaches the first car and impacts the first car. Simultaneously, the operator can be controlling his perspective of the accident scene, such as with joysticks, to zoom in on areas of particular interest, such as the second car as it approaches the first car, to translate from the second car to the first car to view the first car as the second car is approaching, and to rotate for changing the operator's perspective. Operation can be slowed to permit careful observation of the occurrences, can be speeded to obtain a high-speed perspective, and can be stopped to permit still frame investigation. Different accident scenarios can be simulated in accordance with conflicting witness observations.

F

### Train Simulator Application

A train simulator can be implemented in accordance with train simulators presently in use by the railroads and train simulators contracted by the U.S. Department of Transportation. Such simulators can be used for accident evaluation, crew training, and other such purposes. In one such simulator, a motion base with a locomotive cab is used to house the trainees. Terrain along the right of way is presented with moving picture displays recorded on moving picture film. An improvement would generate terrain with an image processor, such as discussed below.

Images can be recorded along the right of way with a still frame camera; such as a photographic camera or a video camera; and stored as static frames in a video database, such as a video disk or video tape database. The image processor can access frames in accordance with the virtual scrolling implementation disclosed herein. The processed images can be directly displayed with CRT monitors simulating windows of the locomotive cab. Alternately, the processed images can be projected with a projection television-type arrangement to provide larger screen images that can be viewed through actual windows of the locomotive cab. One form of projection display is disclosed in Patent No. 4,435,732; which is incorporated herein by reference.

 Different scenarios can be controlled through the capability to branch through the video memory for non-sequential frames in accordance with the virtual scrolling capability. Therefore, different scenarios, such as taking either of two paths at a railroad switch point can be simulated.

Train motion can be simulated, such as by reducing the translational rate through the image processed environment. Train motion can be controlled by the operator, such as with a locomotive throttle control; by an experimenter, such as with an experimenter's panel for controlling the train scenario; or by other means.

This arrangement facilitates ease of database generation, rapid branching capability, and flexible scenario operation.

### Helicopter Training Simulator Application

A helicopter training simulator provides a good example of various simulator applications. Many requirements for a helicopter training simulator exceed those of other aircraft simulators. For example, a helicopter can approach very close to textured objects, such as trees and buildings, while higher flying aircraft may not need close-up simulation of highly textured objects. Also, a helicopter has more discontinuous flight dynamics, being able to cruise at high ground speed and to hover at zero ground speed; while higher flying aircraft do not have such dynamic capability.

A helicopter simulator can be implemented with an image processor providing a background image and with various overlays placed thereon. For example, a background image can be provided portraying background terrain with various overlays placing foreground objects, such as trees and buildings, superimposed on the background image. As is possible with helicopters, a helicopter can fly to a foreground object and be obscured therebehind, such as flying near a tree and hovering behind the tree to avoid detection. The helicopter can then fly around the foreground object and proceed to the next object. The overlay occulting capabilities discussed herein provide occulting between overlay objects. One overlay configuration discussed herein provides for selectively overlaying with a complex image that permits viewing through selected portions of the image, such as viewing of other overlay images and the background image through the spaces between leaves of an overlay tree. The expansion capability discussed herein facilitates approaching an overlay

object, such as a tree, and having the object expand in size as it is approached to simulate 3D-perspective. For example, a tree overlay placed over the background image as a very small overlay image can then be expanded to fill the viewport and to expand beyond filling the viewport as the helicopter approaches the tree and hides behind the tree.

An attack helicopter simulation can be implemented with the system of the present invention, providing high levels of detail and realism. Multiple high detail overlays can be provided; yielding a textured background image with overlayed textured objects, such as trees; and with 3D-perspective. For example, the background image can have textured mountains, hills, waterways, buildings, and meadows. A group of textured images; such as trees, buildings, and ground vehicles; can be overlayed on the background with occulting between images.

In one example, a tree image having textured leaf and branch detail can be independently manipulated; i.e. rotated, translated, scaled, perspective processed, and warped; and overlayed on a textured background image. Initially, the tree can be compressed; appearing as a distant background object. As the helicopter flies into the background toward the tree, other background objects approach, increasing in size, and pass to the sides of the helicopter to portray the effects of motion with perspective. As the helicopter approaches the tree, the tree expands to fill the viewport and to extend beyond the viewport. The helicopter can hover behind the tree; with display of details of leaves and branches in expanded form to high detail. The tree

image can be automatically cropped to an irregular external profile determined by the external leaf and branch patterns. Also, the tree image can be automatically cropped to an internal pattern, permitting the trainee to see through the spaces  
F  
<sup>1</sup><sub>1</sub> ^  
inbetween the leaves and branches; showing the background and the more remote objects occulted by the fine leaf and branch structure of the tree.

As the helicopter "hovers" behind the tree image, motion of the helicopter causes relative motion of the tree image and the occulted background image and object images behind the tree in accordance with 3D-perspective considerations.

### Large Image Applications

Processing of large images will now be illustrated with processing of a Landsat image. A Landsat image can be obtained by satellite scanners and transmitted to a ground station. The ground station can record the image in a database. An operator can examine the image to obtain pertinent information and to select portions for hard copy output. For this example, the image will be considered to be 50,000-pixels square, totaling about 2.5-billion pixels per image, and the image will be considered to be transmitted down at a 2.5-MHz pixel rate, requiring about 1000-seconds for transmission. The transmitted image can be stored in a database, such as implemented with a Winchester disk. Large Winchester disks are readily available having near one billion bytes per disk. Several Winchester drives can be used to store a whole image.

The observer can assess the image, such as to locate particular features of interest. Assessment can include "on the fly" assessment as the image is being loaded and work station assessment after the image has been captured in the database. On the fly assessment provides some limitations on the pixels that can be assessed, based upon the scan pattern and the progress through the transmission. Work station assessment provides greater flexibility because the image is already captured in the database and therefore available for image processing as a whole image. The operator can initially compress the image so that all pixels are compressed to a single viewport image to obtain an overview of the total image. The operator can then selectively roam over the image, translating and rotating to position and

F  
reorient the features to be further examined. The operator can then expand the image relative to the features of interest to view these features with greater resolution. The procedure of compressing the image to provide greater range, positioning and orienting the image to select features, expanding to greater resolution to investigate the features, again compressing the image and positioning and reorienting the image to select other features, again expanding the image to examine the new features, and so forth permits the operator to iteratively investigate portions of the image.

For purposes of illustration, a Landsat satellite will be assumed to scan the earth with 6-different sensors, 4-visual sensors and 2-infrared sensors, generating 6-different images. The 6-images are each digitized into a 50,000-pixel-by-50,000-pixel image for a 2.5-billion pixel image. The 6-images are multiplexed together and transmitted to a ground station in digital signal form. The images are demultiplexed and stored in a database for correction, enhancement, assessment, and recording. The recordings are then disseminated to users.

F  
A block diagram of a Landsat arrangement is shown in Fig 8.  
A Landsat satellite 911A receives images 911I and transmits the images to a ground station 911C to 911H through a <sup>INS</sup> ~~TDSS~~ datalink 911B. The 6-images can be multiplexed together for transmission to the ground station. In the ground station, the images can be demultiplexed with demultiplexer 911C and loaded into a database 911D.

Demultiplexing can include separating of the 6-bands or 6-images into 6-separate image databases, where each image database is processed as individual separate image. Alternately, the 6-images can be packed together so that corresponding pixels of each image are packed into the same pixel word in a single database. For example, each image can be digitized into an 8-bit intensity word per pixel and all 6 8-bit intensity words for a particular pixel can be packed together into the same pixel word. Unpacking of such pixel words can be performed with subsequent processing, such as prior to recording. Alternately, the images can be maintained in packed form, where a particular one of the 6-images is selected for outputting, such as to a recorder; where unpacking is implicit in the selection and outputting operations.

The database can be structured as a 2D-memory map stored on disk memory. The disk memory can be implemented with one or more Winchester disks. For example, a 50,000-pixel square image having 6-bytes per pixel, related to 6-images, requires 2.5-billion triple-byte words. Assuming that an 0.5-billion byte Winchester disk was available, this would require 30 Winchester disks to store the complete image database.

Corrections can be applied to the images, such as radiometric corrections, geometric corrections, and other corrections with device 911G following storage in the database. Alternately, corrections can be applied prior to the images being stored in the database, such as while the images are being transmitted on a pixel-by-pixel basis.

Image enhancement can be provided with image enhancement module 911E, which can be implemented with the spatial processing capability. For example, image enhancement can be provided with kernel filtering; such as for edge enhancement, feature emphasis, low pass filtering, high pass filtering, and other enhancements.

Image assessment module 911H is provided for assessment of the images by an operator prior to recording. Assessment can be used interactively with image enhancement, where an operator iteratively assesses and enhances the image to obtain an image appropriate for recording. Image assessment capability can be provided, such as with the geometric processing capability. For example, an operator can rotate, translate, expand, compress, and warp an image to assess the image. The operator can control various enhancement capabilities to enhance the image based upon the assessment. After the image has been assessed to be appropriate for recording, the image can be recorded on recorder 911F.

Image recording can be performed with film, optical disks, video tape, and other media. Recording on film can be implemented with an arrangement that photographs a CRT monitor having the desired images displayed thereon. Recording on optical disks and video tape can be performed with well-known optical disk and video tape recording devices. Recording can be made of portions of the image to the desired resolution. For example, recordings can include a single frame image of the complete image; a single frame image of a selected portion of the complete image, multiple frame images of the complete image, multiple frame images of selected portions of the complete image,

or other selections. For example, a user may select a particular portion of the earth to be recorded to full resolution with multiple frames and may select a single frame image of the complete image to obtain a recording of the total environment to better visualize the total environment for subsequent assessment of the higher resolution image. Compressed images can be spatially filtered to reduce aliasing and to reduce undersampling.

There are many applications for processing of large images. These include examination of ultra-high resolution images to extract information; such as with remote sensing, surveillance, and cartography (map making) applications; and include processing of very-high resolution images for printing; such as with publishing, business graphics, and tape to film moviemaking applications. Ultra-high resolution images for remote sensing applications can exceed a billion pixels and very-high resolution images for publishing applications can exceed 10-million pixels. These applications involve storing a large image in a database; accessing selected portions of the image from the database; processing the accessed portions of the image; and displaying or recording the processed image.

Satellite pictures can often cover 50,000-pixels by 50,000-pixels involving 2.5-billion pixels. Publishing industry images can cover 15,000-pixels by 15,000-pixels, involving 225-million pixels. Implementation of an integrated circuit memory map for storing this large number of pixels could be excessively expensive. However, an arrangement for storing such large

images in a database; such as on a video disk, video tape, or magnetic disk; and then processing the appropriate portions of this image with the image processor in accordance with the present invention provides important efficiencies. A virtual scrolling arrangement for updating an image from a video disk has been disclosed previously.

Remote sensing and cartography applications are typical of systems requiring ultra-high resolution images. Remote sensing applications include satellite scanners, such as in Landsat applications, and aircraft sensors, such as in reconnaissance applications. Cartography applications includes map making for military, commercial, and consumer applications.

Landsat images are available for military and non-military applications. The Defense Mapping Agency (DMA) processes such images and makes them available for military and non-military use. In the near future, non-military satellites will generate large high detail images for commercial applications. Companies involved in exploration, such as for oil and minerals; in weather, such as TV networks and airlines; in geography, such real estate developers; and government agencies needing information on weather, troop and equipment movements, and other situations will obtain images for analysis.

Geographical images of the type found in World Atlas books; such as topological maps, road maps, natural resource locations, population distributions, and others; will be commercially available on optical disks. Such systems are presently being implemented for direct readout of the image from the optical disk to the display monitor without image processing, such as in a

slide projector. The system of the present invention provides real time interactive image processing to examine such images in an efficient interactive manner.

Printing of very-high resolution images is used in many applications; such as publishing, business graphics, and movie applications. Very-high resolution images can have more than 10-million pixels. Special systems have been developed for exposing very-high resolution images on film, such the Dicomed system for 35mm slides and the Teledyne Camera system for tape to film movie editing. However, such systems have limited image processing capabilities.

The system of the present invention has the capability to accommodate very large images, limited in detail only by the capacity of the database. These images can be geometrically processed, filtered, and otherwise processed for editing and for providing exotic special effects before printing.

Real time interactive geometric processing capability is important for efficient operator interaction. Such capability includes real time interactive rotation, translation, continuous (non-integer) expansion and compression, and warping.

For example, a doctor viewing an X-ray or a cartographer examining a map can visually "pan" over the image to find a feature of interest, "zoom" in closer to see the feature close up, and rotate the image to change its orientation. Also, warping provides the capability to remove distortions, such as lens anomalies; to remove 3D-effects, such as 3D-perspective compressing the image at more remote locations; and to match up different images for comparison.

### Image Processing Workstation Application

An image processing workstation can be used in many applications; such as processing of large images, industrial arts, graphic arts, business graphics, and others. Such a workstation will be illustrated with a discussion of a Landsat image processing workstation with reference to Fig 11B.

An image processing workstation can be configured as a stand-alone workstation for viewing a large Landsat image.

Viewing of large images (i. e. 368-million pixels) can be performed to maximum resolution for assessing details of the image, at maximum compression for assessing the whole image simultaneously on the CRT monitor, and at any level of expansion/compression <sup>1/1</sup> <sub>1</sub> between these limits for the desired level of detail consistent with the range and resolution of the CRT monitor. The operator can translate over the image for viewing different portions of the image and can rotate the image for viewing different orientations of the image. All operations can be performed in real time, updating the CRT monitor at a 30-frame per second rate for smooth continuous motion. Expansion and compression capability range from the whole image being compressed for simultaneous viewing on the CRT monitor through a highly expanded image to maximum resolution for viewing details on the CRT monitor.

The workstation can be configured as a self-contained stand-alone system, housed in an ergonomically-designed table-type enclosure. The operator can be seated at the terminal 910C for convenient viewing of a color CRT monitor and an alphanumeric terminal. Joystick controls can provide for rotating,

translating, expanding, and compressing the image. A cursor can control the center of rotation and the center of expansion and compression. A keyboard can provide communication between the operator and the supervisory processor 910B. The supervisory processor 910B can provide communication to a host computer 910A. Off-line operation of the workstation relieves processing and communication loading of the host computer 910A. A complete image, such as a 400-million pixel image, can be stored on the self-contained Winchester disk memory 910D for off-line image processing operations.

The image processor can be implemented as a pipeline processor, as shown in Fig 9B. The image memory 910I is loaded from the disk memory 910D and 910E; the geometric processor 910H processes the information in image memory 910I for rotation, translation, expansion, and compression; the spatial processor 910J processes the geometrically processed information for anti-aliasing, smoothing, and interpolation; and the CRT interface 910K converts the digital geometrically and spatially processed information into video signals for exciting the CRT monitor 910L. A supervisory processor 910B is provided for control of system operations, anticipatory memory management for disk accesses, and communication with the host computer 910A.

Image information from host computer 910A or disk 910D can be loaded with buffer memory 910F for compression processing with compression processor 910G. Compressed information can be loaded into image memory 910I for geometric processing or can be loaded onto disk memory 910D for storage.

F  
F

Images are loaded onto the disk memory 910D. The image processor accesses portions of the image from the disk memory 910D for loading into the image memory 910I. It then dynamically manipulates the image in image memory 910I in real time in response to operator commands to for viewing of the image. If dynamic operations exceed the boundaries of the image stored in image memory, such as translating towards a boundary of the image memory or compressing beyond the information contained in image memory, additional pixels are accessed from disk memory 910D to support such operations. Image processing operations within the image memory occur instantaneously; within a frame period. Image processing operations that exceed the boundaries of the image memory invoke disk memory accesses to re-load image memory. Anticipatory disk accesses can make disk operations almost transparent to the operator.

The CRT monitor 910L is a color monitor having extended medium resolution (about 1/3-million pixels), expandable to high resolution (about 1.3-million pixels).

Expansion and compression capability provides almost instantaneous expansion and compression between maximum and minimum display size without interrupting real time operation and almost unconstrained by disk access considerations.

Two representative configurations are specified in the IMAGE PROCESSING WORKSTATION TABLE.

IMAGE PROCESSING WORKSTATION TABLE

Configuration	Configuration-1	Configuration-2
CRT Resolution		
Lines	480	1100
Pixel/line	700	1200
Total pixels	0.336-million	1.32-million
Real time operation		
Updates	Yes 30-frames/sec	Yes 30-frames/sec
Pixel/second	10-million	40-million
Image processing		
Rotation,	Yes	Yes
Continous	Yes	Yes
Resolution	0.2-degrees	0.2-degrees
Max rate	360-degrees/sec	360-degrees/sec
Translation	Yes	Yes
Continuous	Yes	Yes
Resolution	1-pixel	1-pixel
Max rate	10,000-pixels/sec	20,000-pixels/sec
Expansion	Yes	Yes
Continuous, non-integer	Yes	Yes
Resolution	1-pixel	1-pixel
Max rate		
< Double size	1/30-second	1/30-second
> Double size	About 5-seconds	About 5-seconds
Pixel replication	No	No
Interpolation	Yes	Yes
Anti-aliasing	Yes	Yes
Compression		
Continuous, non-integer	Yes	Yes
Resolution	1-pixel	1-pixel
Max rate		
< Half size	1/30-second	1/30-second
> Half size	About 5-seconds	About 5-seconds
Undersampling	No	No
Anti-aliasing	Yes	Yes
Kernel	9-pixels	9-pixels
Weights	Programmable	Programmable
Kernels/second	10-million	40-million
Max compression		
Full image (pixels)	0.336-million	1.32-million
Max expansion		
Full image (pixels)	67-million	268-million
Overlays		
Cursor	Yes	Yes
Symbols	Yes	Yes
Database, Winchester disk		
Total image	Yes	Yes
Latency	5-seconds	5-seconds

### Arcade Game Application

An arcade game application is representative of many different types of video disk image processing applications; including interactive entertainment applications, simulation applications, and training applications. This application will be discussed in the context of a video disk-based system as being illustrative of other database memory arrangements.

The system of the present invention provides high quality imagery of a video system together with the freedom of motion of the conventional graphic video game systems. It provides unlimited freedom of motion, consistent with the pixel resolution of a video disk and CRT monitor. The images can move from any pixel to any of the adjacent pixels in any direction, yet require a significantly smaller database of image frames and a significantly smaller number of disk accesses than conventional video disk based game approaches. This is achieved by storing static non-motion frames on the video disk and introducing motion with a geometric processor. Therefore, multiple redundant frames required by conventional video disk based systems to provide limited motion and the disk accesses necessary to read the multiple redundant frames is overcome by the present image processor. The geometric processor can modify a static frame or an overlapping group of static frames to provide dynamic effects with high resolution motion.

The geometric processor can perform various motion-related operations; including translation, rotation, expansion, and contraction. In an airborne game application, such as a video bombing game; high detailed terrain can be provided covering a

wide range of area to be flown over. The operator can fly the plane, which rotates and translates over the ground terrain under pilot-player control; providing the appearance that the operator is actually flying an aircraft over a high degree of freedom ground path. The operator can change altitude, where increases in altitude cause the ground terrain to be compressed and decreases in altitude cause the ground terrain to be expanded as the terrain is being rotated and translated under operator control. Further visual effects include range-related image compression, where for example an aircraft banking <sup>maneuver</sup> ~~manuever~~ shows terrain at a greater range, such as near the horizon, to be more compressed and terrain at a lesser range, such as under the aircraft, to be less compressed. This provides a 3-dimensional perspective for depth perception and distance perception.

A database of images can be stored on a video disk. Sufficient information can be stored on a small video disk to permit an operator to control an unlimited number of scenarios. Trillions of game scenarios can be provided with imagery stored on a small video disk. Hundreds of trillions of different image frames can be provided from a limited amount of compressed images stored on a small video disk. An operator can control the game scenario to follow a virtually unlimited number of paths. Also, video disk access rates are relatively infrequent because a small number of frames accessed from video disk can be processed to provide multitudes of displayed dynamic images, derived by geometrically processing the single accessed frame. This further reduces dependence on video disk operation; where a relatively

simple, low capacity, slow access rate, and less rugged video disk drive can be used.

F Advantages are derived by accessing static non-redundant visual information from a video disk and then processing the static stored information to provide dynamic visual information under interactive operator control. A single frame of static imagery can be geometrically processed to generate multitudes of frames of dynamic imagery. Unequal image compression and image expansion simulates the more sophisticated 3D-perspective effects.

Additional capability includes a high detail moving background and high detail moving objects in the foreground occulting or obscuring the more remote background imagery behind the foreground moving objects. 3D-perspective provides the effect of slower motion for a more remote background and faster motion for the less remote foreground imagery to enhance realism.

An aircraft example will now be provided to illustrate capabilities. The background imagery can include ground patterns consisting of textured frames of ground patterns; such as foliage, waterways, mountains, farmland, and others. This textured background can be generated with static aerial photographs, such as used for map making. Static images stored on video disk are accessed and dynamically processed to provide rotated, translated, compressed, and expanded dynamic versions of the static pre-recorded images to simulate motion. As the operator flies a simulated aircraft over the terrain, the static pre-recorded images are accessed from database memory and are rotated and translated to provide the appearance of a moving

background. As the operator increases or decreases the aircraft altitude, the background terrain is expanded or compressed respectively to provide the appearance of increased or decreased altitude. The field of view causes the amount of compressed terrain seen on the screen to be increased as altitude increases and to be decreased as the altitude decreases, consistent with viewing terrain through an aircraft window. The portion of the imagery at greater distances is compressed more than the portion of the imagery at nearer distances with 3D-perspective. Also, the portion of the imagery at greater distances moves more slowly than the portion of the imagery at nearer distances in accordance with 3D-perspective. Motion is continuous in real time at the frame rate, motion is unlimited in direction, motion changes in direction are unlimited, and scenario duration is unlimited. An aircraft overlay can be superimposed on the background imagery to enhance the effect of flying a plane. Auxiliary images, such as bomb explosions superimposed on the background terrain, facilitate game effects. Foreground images, such as aircraft and bomb explosion images, overlay and obscure other images. For example, an aircraft image obscures all more remote images, such as background terrain images and bomb explosion images, and bomb explosion images obscure all more remote images, such as background terrain images.

A hero and dragon game example of operation will now be provided to further illustrate the capabilities. The background imagery can include dungeon effects consisting of textured stonework, fixtures, furniture, and others. As discussed above,

this textured background need not be generated with animation but can be generated with static photographs, such as generated on television or movie sets. Static images stored on video disk are accessed and dynamically processed to provide rotated, translated, compressed, and expanded dynamic versions of the static pre-recorded images to simulate motion. As the operator moves the hero, the static pre-recorded images are accessed from database memory and are rotated and translated to provide the appearance of a moving background. As the operator increases or decreases the distance from the background, background imagery is expanded or compressed respectively to provide the appearance of increased or decreased distance from the background imagery. The field of view causes the amount of compressed terrain seen on the screen to be increased as distance increases and to be decreased as the distance decreases, consistent with viewing the background through a window. The portion of the imagery at greater distances is compressed more than the portion of the imagery at nearer distances with 3D-perspective. Also, the portion of the imagery at greater distances moves more slowly than the portion of the imagery at nearer distances in accordance with 3D-perspective. Motion is continuous in real time at the frame rate, motion is unlimited in direction, motion changes in direction are unlimited, and scenario duration is unlimited. A hero overlay can be superimposed on the background imagery to enhance the effect of controlling the hero. Auxiliary images, such as <sup>VILLAINS</sup> superimposed on the background, can be provided to facilitate game effects. Foreground images overlay and obscure other images. For example, a hero image obscures all

*F*  
*F*  
more remote images, such as background images and <sup>VILLIAN</sup> villian images,  
and <sup>VILLIAN</sup> villian images obscure all more remote images, such as  
background images.

NON-DISPLAY APPLICATIONS

### General Description

Various applications of the system of the present application are non-display applications; such as automatic pattern recognition, robotics, and artificial intelligence applications. These applications are similar to display applications in that they need processed images. These applications are different from display applications in that evaluation of the processed image is performed digitally, not visually by an operator; typically involving comparison of an input sensed image with a stored image. It is often necessary to register an acquired image and a reference image so that they can be overlayed to match up corresponding elements for comparison. Registration can be performed by rotating, translating, expanding, compressing, and warping one or both images to bring corresponding features into registration. Correlation may then be performed by comparing the now-registered images. The geometric processor provides the capability for registering the images and the spatial processor provides the capability for comparing the registered images.

The high pixel rate image processing discussed herein increases throughput in non-display applications. For example, a pattern recognition system is often rated based upon the number of images that can be processed per second, the number of different types of filtering operations that can be performed on an image per second, and the pixel per second throughput of the system. Also, the overlay capability for display applications discussed herein facilitates multiple channel pattern recognition-type capability in non-display applications; such as

for parallel pipeline processing. Therefore, the same capability that facilitates smooth real time operation and overlays for display applications, as discussed herein, provides comparable advantages for non-display applications.

Robotics relates to industrial automation applications. An image can be captured by a video camera, digitized, and loaded into image memory. The image can be geometrically processed to register with a reference image, spatially filtered to enhance recognizable features, and used to control a machine to perform the required function. For example, on an automotive assembly line, a welder can be controlled to weld together two irregular parts based upon "visual" control of the welder.

Visual guidance relates to a form of robotics for a robot that is free to move from a fixed position. The image processor discussed herein can provide such visual guidance. Such machines include guided weapons, satellites, aircraft, land vehicles, and even Star Wars R2D2 type robots. Guided weapons, such as cruise missiles and intelligent bombs, can be guided using mid-course and terminal visual guidance systems. Satellites can be positioned and oriented and spacecraft can be docked using visual control systems. Land vehicles can be guided using a visual guidance system; automatically following a roadway and detecting and circumventing obstacles.

Remote sensing and surveillance applications relate to processing of images derived from scanners, such as satellite and airborne scanners. The image processor discussed herein provides registration and filtering of images for display and for

automatic processing. Registration can be used to matchup mosaics; such as warping one map mosaic to match up with an adjacent map mosaic; for composing a large consistent seamless map from a group of mosaics. Registration can also be used to compensate for distortions, such as with warping to remove range perspective effects and camera lens effects. Geometric processing permits an operator to investigate an image; such as by roaming over the image with translation, rotating to orient a feature of the image, expanding to see the feature in greater detail, and compressing to look for other features. Filtering can be used to enhance the image; such ~~to~~ to enhance edges, patterns, and features. The registered and filtered image can be displayed to an operator, such as for surveillance, or can be processed with an automatic pattern recognition system or artificial intelligence system.

F

### Pattern Recognition

The present image processing invention can be used in various applications needing automatic pattern recognition capability. These include automatic inspection systems, homing systems, remote control systems, and others. The ability to rotate, translate, and scale an image facilitates comparing a plurality of images, such as comparing an acquired image and a stored image. Error signals can be derived indicative of the alignment or registration errors between different images, such as an acquired image and a stored image. A closed loop servo can be used to drive images to register with each other, such as driving a stored image to align with an acquired image. Registration may be considered to be a nulling of spatial error signals. Driving functions for registration of images can be measured as a function of spatial positions of the images and can be used to register images.

The system of the present invention has two primary features for pattern recognition. One feature is the geometric processor for rotating, translating, expanding/compressing, and warping for providing registration between 2-images for comparison. Another feature is the spatial filter, which performs functions such as edge enhancement and feature detection. Both of these features can be used separately or together to facilitate pattern recognition. Other features of the present invention can further enhance pattern recognition.

There are various types of edge detection, such as for pattern recognition and image enhancement. These include convolution and shift and subtraction.

Convolution can be provided with weights that erode edges, expand edges, etc. Convolution with higher latitude can also be provided. The convolution approach can process kernel intensities to provide a new output intensities for a pixel. This can be implemented by using the input to the weight table, being the kernel pixel intensities, and the output of the weight table, being the new intensity for the center pixel. For example, if the intensities are equal, a zero-intensity output is provided and if there is a change in intensity an appropriate non-zero intensity is output. This is effectively a differentiation form of table lookup.

Another way to detect edges is to shift an image, such as one pixel to the right, and then subtract the shifted image from the unshifted image.

Processing of error signals will now be discussed. Error signals can be derived that are indicative of differences in position, orientation, scale, and other spatial characteristics between different images. For example, features of acquired and reference images can be identified and the difference in the X-position and Y-position of the features in the acquired image and the reference image can be measured as error signals.

Identification of various image features and measurement of alignment errors related thereto can be used to determine position, orientation, scale, and other characteristics of an acquired image.

Nulling of alignment errors can be accomplished by nulling each feature as the error conditions are determined or, alternately, by nulling the total image after the errors in all pertinent features are determined. The former nulling approach may be considered to be a sequential nulling configuration; such as identifying a registration error associated with a first feature and nulling this first feature error, then determining a registration error associated with a second feature and nulling this second feature error while keeping the first feature error nulled, then determining a registration error associated with a third feature and nulling this third feature error while keeping the first two feature errors nulled, and so forth. The latter nulling approach may be considered to be a parallel nulling configuration; such as measuring errors associated with a plurality of features and solving mathematical equations, such as parametric equations, for determining each error as a function of the positions of a plurality of features.

Alternately, a correlation or convolution spatial filtering nulling configuration can be implemented. For example, an acquired image can be convolved with a reference image to determine the misalignment therebetween.

One nulling configuration can implement a driving function technique; where a driving function can be used to rotate, translate, scale, warp, and otherwise process one or more images to bring the images into alignment. This may be particularly appropriate for a configuration that can determine error values explicitly, such as by measuring the relative X-position and Y-

position of identifiable features of the images.

An alternate nulling configuration can implement a scanning technique, where one or both images are scanned in a scan pattern and the alignment errors for different spatial conditions are measured to determine errors for those conditions. Scanning can be implemented in multiple degrees of freedom; such as 2-dimensions of translation, 3-dimensions of rotation, and 1-dimension of scaling. Additional degrees of freedom can be implemented, such as warping of an image and range variable scaling of an image. Scans can be in the form of a raster scan, a Palmer scan, a circular scan, and other forms of scans.

Operation may be enhanced by selecting different resolutions for different modes of operation. For example, reduced resolution can provide more rapid image processing and improved utilization of image processing resources. Reduced resolution image processing can be performed over a larger image area, such as for coarse alignment, and increased resolution image processing can be performed over a more limited image area, such as for fine alignment of coarse aligned image features. For example, coarse alignment can cover a greater spatial range, because the images may initially be widely misaligned, and fine alignment can cover a lower spatial range, because the images may have already been coarse aligned with a coarse alignment operation. Therefore, coarse alignment may require greater spatial range, but to lesser spatial resolution, and fine alignment may require smaller spatial range, but to greater spatial resolution.

In one configuration; a coarse alignment mode may be implemented to rapidly bring images into course alignment, a fine alignment mode may be implemented to more precisely align the images, and a measurement mode may be implemented for more precise measurement of image features and nulling of alignment errors. Each of a plurality of modes can be implemented with different resolutions, consistent with the operations to be performed. For example, a coarse alignment mode can be implemented to coarse resolution, a fine alignment mode can be implemented to finer resolution, and a measurement mode can be implemented to still finer resolution.

### Inspection Applications

F Automatic inspection applications will now be discussed as illustrative of various image processing applications for measuring, locating, correcting, and otherwise operating of images. For example, an assembly line having parts traveling down the line can have parts misaligned to the assembly line. For this example, it is desirable to determine dimensions of a part and location of a feature of the part. An image processor can have a reference image of the part stored in memory map form. A video camera can acquire an actual image of the part on the assembly line. The acquired image can be compared with the reference image to determine error conditions. A nulling processor can translate, rotate, and expand or compress the reference image or the acquired image to null errors and to align the reference image and the acquired image.

The reference image can be driven with driving functions to track movement of the acquired image as it proceeds down the assembly line. Driving functions can be predefined driving functions associated with a known speed of the assembly line, can be automatically derived driving functions determined by a change in the error conditions between the two images from frame to frame, or can be a combination of pre-determined assembly line motion and a vernier automatic alignment to compensate for variations in assembly line motion.

After part alignment has been achieved, errors between important features can be measured. For example, alignment of selected features of the images can be used to initially align the images and measurement of errors between images that are

already aligned can be determined for measurement of tolerances of the part on the production line; i.e., alignment of corners of a cube can be used for initial alignment and measurement of errors of other features of the cube can be used to determine tolerances.

COMPUTER PORT TABLE

<u>BIT</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>PORT-A</u>	<u>CONTROL</u>	<u>PORT</u>
					<u>NOTES</u>
0	DOA0	TEST PULSE-1			
1	DOA1	JOYSTICK SELECT-1			
2	DOA2	JOYSTICK SELECT-0			
3	DOA3			HIGH IS UNBLANKED, LOW IS BLANKED	
4	DOA4	UNBLANK		SEQUENTIAL LOAD OF IMAGE MEMORY	
5	DOA5	SEQUENTIAL-LOAD		BRACKETED BY FRAME BLANKING CFS	
6	DOA6	LOAD-BAR/RUN		BRACKETED BY PBO AND PCO	
7	DOA7	DATA STROBE			
0	DIA0	FRAME SYNC		VERTICAL FIELD BLANKING PULSE	
1	DIA1				
2	DIA2	LINE SYNC		HORIZONTAL LINE BLANKING PULSE	
3	DIA3				
4	DIA4	FRAME-1			
5	DIA5				
6	DIA6				
7	DIA7				

#### SEQUENTIAL CONTROL OF IMAGE LOADING

- (1) RESET DOA5
- (2) LOAD X+Y START ADDRESSES, (PIXEL ADDRESS TIMES 8)
- (3) LOAD SLOPES TIMES 256
- (4) SET DOA5
- (5) GENERATE A SEQUENCE OF STROBED DATA OUTPUTS ALONG LINE
- (6) ADDRESS COUNTER WILL AUTOMATICALLY ADVANCE
- (7) DOA5 must be set before strobing data into memory independent of whether one or a sequence of strobes is to be generated.

<u>BIT</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>ADDRESS/DATA PORT</u>
0	DOB0	LSB	
1	DOB1	!	
2	DOB2	!	
3	DOB3	!	
4	DOB4	!	
5	DOB5	!	
6	DOB6	!	
7	DOB7	MSB	
0	DIB0	LSB	
1	DIB1	!	
2	DIB2	!	
3	DIB3	!	
4	DIB4	!	
5	DIB5	!	
6	DIB6	!	
7	DIB7	MSB	

<u>BIT</u>	<u>NAME</u>	<u>FUNCTION</u>	<u>DESTINATION</u>	<u>SELECT PORT</u>	<u>NOTES</u>
0	DOC0	LSB			
1	DOC1	!			
2	DOC2	!			
3	DOC3	!			
4	DOC4	!			
5	DOC5	!			
6	DOC6	!			
7	DOC7	MSB			
0	DIC0	LSB			
1	DIC1	!			
2	DIC2	!			
3	DIC3	!			
4	DIC4	!			
5	DIC5	!			
6	DIC6	!			
7	DIC7	MSB			

<u>P</u>	<u>B7</u>	<u>B6</u>	<u>B5</u>	<u>B4</u>	<u>B3</u>	<u>B2</u>	<u>B1</u>	<u>B0</u>	<u>REGISTER</u>	<u>SELECT</u>	<u>ASSIGNMENTS</u>
0	0	0	0	0	0	0	0	0	0	<u>B7</u>	<u>B6</u>
1	0	0	0	0	0	0	0	1	0	<u>B5</u>	<u>B4</u>
2	0	0	0	0	0	0	1	0	0	<u>B3</u>	<u>B2</u>
.	.	.	.	.	.	.	.	.	.	<u>B1</u>	<u>B0</u>
127	0	0	0	0	1	1	1	1	0		
128	0	0	0	1	0	0	0	0	0		

129 0 0 0 1 0 0 0 1 0 0 1  
130 0 0 0 1 0 0 0 1 0 0 1

· · · · · · · · · · · ·

240 1 1 1 0 0 0 0 0 0 0 1  
241 1 1 1 0 0 0 0 1 0 0 0  
242 1 1 1 0 0 0 1 0 0 0 0  
243 1 1 1 0 0 0 1 1 0 0 0  
244 1 1 1 1 0 0 1 0 0 0 0  
245 1 1 1 1 0 0 1 0 1 0 0  
246 1 1 1 1 0 0 1 1 0 0 0  
247 1 1 1 1 0 1 0 1 1 0 0  
248 1 1 1 1 1 0 0 0 0 0 0  
249 1 1 1 1 1 0 0 1 0 0 0  
250 1 1 1 1 1 0 1 0 1 0 0  
251 1 1 1 1 1 0 1 1 0 1 1  
252 1 1 1 1 1 1 0 0 0 0 0  
253 1 1 1 1 1 1 0 1 0 1 0  
254 1 1 1 1 1 1 1 1 1 1 1  
255 1 1 1 1 1 1 1 1 1 1 1

XR-DELTA LSH 0 0 0 0 0 0 0 0 0 0 0  
YR-DELTA LSH 0 0 0 0 0 0 0 0 0 0 0  
XP-DELTA MSH 0 0 0 0 0 0 0 0 0 0 0  
XP-DELTA LSH 0 0 0 0 0 0 0 0 0 0 0  
YP-DELTA LSH 0 0 0 0 0 0 0 0 0 0 0  
YP-DELTA MSH 0 0 0 0 0 0 0 0 0 0 0  
XR-ADDRESS MSH 0 0 0 0 0 0 0 0 0 0 0  
XR-ADDRESS LSH 0 0 0 0 0 0 0 0 0 0 0  
XR-DELTA MSH 0 0 0 0 0 0 0 0 0 0 0  
YR-ADDRESS MSH 0 0 0 0 0 0 0 0 0 0 0  
YR-ADDRESS LSH 0 0 0 0 0 0 0 0 0 0 0  
YR-DELTA MSH 0 0 0 0 0 0 0 0 0 0 0  
DATA D7 D6 D5 D4 D3 D2 D1 D0  
WEIGHTS, LS BYTE (W2) W7 W6 W5 W4 W3 W2 W1 W0  
WEIGHTS, MS NIBBLE(W3) 0 0 0 W11 W10 W9 W8  
SPARE

NOTES

1. Frame sync and CP-bar must be synchronized
2. Gated clock.
3. First line sync per frame is under the frame sync envelope. Therefore, U20D-1 generates second line sync as first pulse.
4. First line sync under frame sync envelope loads pixel registers.

CXPSM	XP-DELTA MSB CLOCK
CXPSP	XP-DELTA LSB CLOCK
CYPSM	YP-DELTA MSB CLOCK
CYPSP	YP-DELTA LSB CLOCK
CXRSM	XR-DELTA MSB CLOCK
CXRSL	XR-DELTA LSB CLOCK
CYRSM	YR-DELTA MSB CLOCK
CYRSL	YR-DELTA LSB CLOCK
CD	DATA STROBE
CXRM	XR-ADDRESS MSH CLOCK
CXRL	XR-ADDRESS LSH CLOCK
CXPM	XP-ADDRESS MSH CLOCK
CXPL	XP-ADDRESS LSH CLOCK
CYRM	YR-ADDRESS MSH CLOCK
CYRL	YR-ADDRESS LSH CLOCK
CYPM	YP-ADDRESS MSH CLOCK
CYPL	YP-ADDRESS LSH CLOCK

CABLE CONNECTION TABLE

CABLE-1 BM1,2/BM1 (C1)

<u>PIN</u>	<u>SIGNAL</u>	<u>SIGNAL DESCRIPTION</u>	<u>SOURCE</u>	<u>DESTINATION</u>
1	GROUND	GROUND		
2	YA0	MEMORY ADDRESS, BIT-Y0	BL1-U9E-10	BM1,2-U19B-1
3	GROUND	GROUND		
4	YA1	MEMORY ADDRESS, BIT-Y1	BL1-U9E-12	BM1,2-U19B-2
5	GROUND	GROUND		
6	YA2	MEMORY ADDRESS, BIT-Y2	BL1-U9E-15	BM1,2-U19B-3
7	GROUND	GROUND		
8	YA3	MEMORY ADDRESS, BIT-Y3	BL1-U8E-2	BM1,2-U19A-2
9	GROUND	GROUND		
10	YA4	MEMORY ADDRESS, BIT-Y4	BL1-U8E-5	BM1,2-U19A-4
11	GROUND	GROUND		
12	YA5	MEMORY ADDRESS, BIT-Y5	BL1-U8E-7	BM1,2-U19A-6
13	GROUND	GROUND		
14	YA6	MEMORY ADDRESS, BIT-Y6	BL1-U8E-10	BM1,2-U19A-10
15	GROUND	GROUND		
16	YA7	MEMORY ADDRESS, BIT-Y7	BL1-U8E-12	BM1,2-U19A-12
17	GROUND	GROUND		
18	YA8	MEMORY ADDRESS, BIT-Y8	BL1-U8E-15	BM1,2-U19A-14
19	GROUND	GROUND		
20	XA0	MEMORY ADDRESS, BIT-X0	BL1-U9D-10	BM1,2-U19C-1
21	GROUND	GROUND		
22	XA1	MEMORY ADDRESS, BIT-X1	BL1-U9D-12	BM1,2-U19C-2
23	GROUND	GROUND		
24	XA2	MEMORY ADDRESS, BIT-X2	BL1-U9D-15	BM1,2-U19C-3
25	GROUND	GROUND		

26	XA3	MEMORY ADDRESS,	BIT-X3	(BM2)	BL1-U8D-2
27	GROUND	GROUND			
28	XA3-BAR	MEMORY ADDRESS,	BIT-X3-BAR	(BM1)	BL1-U18D-4
29	GROUND	GROUND			
30	XA4	MEMORY ADDRESS,	BIT-X4		BL1-U8D-5
31	GROUND	GROUND			
32	XA5	MEMORY ADDRESS,	BIT-X5		BL1-U8D-7
33	GROUND	GROUND			
34	XA6	MEMORY ADDRESS,	BIT-X6		BL1-U8D-10
35	GROUND	GROUND			
36	XA7	MEMORY ADDRESS,	BIT-X7		BL1-U8D-12
37	GROUND	GROUND			
38	XA8	MEMORY ADDRESS,	BIT-X8		BL1-U8D-15
39	GROUND	GROUND			
40	W1-BAR	MEMORY READ/WRITE	(LOW => WRITE)		BL1-U22C-11
41	GROUND	GROUND			
42					
43	GROUND	GROUND			
44					
45	GROUND	GROUND			
46					
47	GROUND	GROUND			
48					
49	GROUND	GROUND			
50					

CABLE-II BM1,2/BL1/BB1 (C2)

<u>PIN</u>	<u>SIGNAL</u>	<u>SIGNAL DESCRIPTION</u>	<u>SOURCE</u>	<u>DESTINATION</u>
1	GROUND	GROUND	BL1-C6-9	BM1,2-U17A,B,C,D-4
2	M10	MEMORY DATA INPUT, BIT-0 (PB0)	BL1-C6-9	BM1,2-U17A,B,C,D-4
3	GROUND	GROUND	BL1-C6-22	BM1,2-U17A,B,C,D-7
4	M11	MEMORY DATA INPUT, BIT-1 (PB1)	BL1-C6-22	BM1,2-U17A,B,C,D-7
5	GROUND	GROUND	BL1-C6-10	BM1,2-U17A,B,C,D-9
6	M12	MEMORY DATA INPUT, BIT-2 (PB2)	BL1-C6-10	BM1,2-U17A,B,C,D-9
7	GROUND	GROUND	BL1-C6-23	BM1,2-U17A,B,C,D-12
8	M13	MEMORY DATA INPUT, BIT-3 (PB3)	BL1-C6-23	BM1,2-U17A,B,C,D-12
9	GROUND	GROUND	BL1-C6-11	BM1,2-U18A,B,C,D-4
10	M14	MEMORY DATA INPUT, BIT-4 (PB4)	BL1-C6-11	BM1,2-U18A,B,C,D-4
11	GROUND	GROUND	BL1-C6-24	BM1,2-U18A,B,C,D-7
12	M15	MEMORY DATA INPUT, BIT-5 (PB5)	BL1-C6-24	BM1,2-U18A,B,C,D-7
13	GROUND	GROUND	BL1-C6-12	BM1,2-U18A,B,C,D-9
14	M16	MEMORY DATA INPUT, BIT-6 (PB6)	BL1-C6-12	BM1,2-U18A,B,C,D-9
15	GROUND	GROUND	BL1-C6-25	BM1,2-U18A,B,C,D-12
16	M17	MEMORY DATA INPUT, BIT-7 (PB7)	BL1-C6-25	BM1,2-U18A,B,C,D-12
17	GROUND	GROUND	BIT-0 (GREEN)	BM1,2-U17A,B,C,D-2 BB1-U...;C4-22
18	M00	MEMORY DATA OUTPUT,	BIT-0 (GREEN)	BM1,2-U17A,B,C,D-2 BB1-U...;C4-22
19	GROUND	GROUND	(GREEN)	BM1,2-U17A,B,C,D-5 BB1-U...;C4-24
20	M01	MEMORY DATA OUTPUT,	BIT-1 (GREEN)	BM1,2-U17A,B,C,D-5 BB1-U...;C4-24
21	GROUND	GROUND	(GREEN)	BM1,2-U17A,B,C,D-11 BB1-U...;C4-26
22	M02	MEMORY DATA OUTPUT,	BIT-2 (GREEN)	BM1,2-U17A,B,C,D-11 BB1-U...;C4-26
23	GROUND	GROUND	(RED)	BM1,2-U17A,B,C,D-14 BB1-U...;C4-28
24	M03	MEMORY DATA OUTPUT,	BIT-3 (RED)	BM1,2-U17A,B,C,D-14 BB1-U...;C4-28
25	GROUND	GROUND		

26	MO4	MEMORY DATA OUTPUT,	BIT-4 (RED)	BM1, 2-U18A, B, C, D-2	BB1 ;C4-30
27	GROUND	GROUND	BIT-5 (BLUE)	BM1, 2-U18A, B, C, D-5	BB1 ;C4-32
28	MO5	MEMORY DATA OUTPUT,	BIT-6 (BLUE)	BM1, 2-U18A, B, C, D-11	BB1 ;C4-34
29	GROUND	GROUND	BIT-7 (SPARE)	BM1, 2-U18A, B, C, D-14	BB1 ;C4-36
30	MO6	MEMORY DATA OUTPUT,	(RED)	BM1, 2-U18A, B, C, D-15	
31	GROUND	GROUND	(BLUE)	BM1, 2-U17A, B, C, D-15	
32	MO7	MEMORY DATA OUTPUT,	U19C-6	BB1, U23C-1, 15	
33	GROUND	GROUND	BL1-U21E-6	BB1, U23D-1, 15	
34	DIEN-BAR MEMORY READ/WRITE				
35	GROUND	GROUND			
36	GROUND	GROUND			
37	GROUND	GROUND			
38	GROUND	GROUND			
39	GROUND	GROUND			
40	GROUND	GROUND			
41	GROUND	GROUND			
42	GROUND	GROUND			
43	GROUND	GROUND			
44	GROUND	GROUND			
45	GROUND	GROUND			
46	GROUND	GROUND			
47	GROUND	GROUND			
48	GROUND	GROUND			
49	GROUND	GROUND			
50					

CABLE-III    BR1/BIL1/BB1    (C3)

<u>PIN</u>	<u>SIGNAL</u>	<u>DESCRIPTION</u>	<u>SOURCE</u>	<u>DESTINATION</u>
1	GROUND	GROUND		BB1-U22C-10
2	YW0	WEIGHT ADDRESS		BB1-U22C-13
3	GROUND	GROUND		BB1-U22C-13
4	YW1	WEIGHT ADDRESS		BB1-U22C-6
5	GROUND	GROUND		BB1-U22C-3
6	YW2	WEIGHT ADDRESS		BB1-U22D-3
7	GROUND	GROUND		BB1-U22D-3
8	XW0	WEIGHT ADDRESS		BB1-U22D-6
9	GROUND	GROUND		BB1-U22D-13
10	XW1	WEIGHT ADDRESS		BB1-U22D-10
11	GROUND	GROUND		BB1-U7B-4
12	XW2	WEIGHT ADDRESS		
13	GROUND	GROUND		
14	YA0	WEIGHT ADDRESS		
15	GROUND	GROUND		
16	YA1	WEIGHT ADDRESS		
17	GROUND	GROUND		
18	DOA6			
19	GROUND	GROUND		
20	GROUND	GROUND		
21	CPG	GATED CLOCK PULSE	BL1-U21D-8	BB1-U7A, U6A, U5A-2
22	GROUND	GROUND		
23	GROUND	GROUND		
24	GROUND	GROUND		
25	GROUND	GROUND		

26 GROUND GROUND  
27 GROUND GROUND  
28 GROUND GROUND  
29 GROUND GROUND  
30 GROUND GROUND  
31 GROUND GROUND  
32 GROUND GROUND  
33 GROUND GROUND  
34 GROUND GROUND  
35 GROUND GROUND  
36 GROUND GROUND  
37 GROUND GROUND  
38 GROUND GROUND  
39 GROUND GROUND  
40 GROUND GROUND  
41 GROUND GROUND  
42 GROUND GROUND  
43 GROUND GROUND  
44 GROUND GROUND  
45 GROUND GROUND  
46 GROUND GROUND  
47 GROUND GROUND  
48 W2-BAR WEIGHT RAM READ/WRITE (LOW=>WRITE) BL1-U22C-6 BB1-U21E, U22E-10  
49 GROUND GROUND  
50 W3-BAR WEIGHT RAM READ/WRITE (LOW=>WRITE) BL1-U22C-8 BB1-U23E-10

CABLE-IV    BR1/BL1/BB1    (C4)

<u>PIN</u>	<u>SIGNAL</u>	<u>SIGNAL DESCRIPTION</u>	<u>SOURCE</u>	<u>DESTINATION</u>
1	GROUND	GROUND	BR1	BL1-U22E-11, C5-2
2	FRAME SYNC	(CFS)	BR1	BL1-U22E-13, C5-3
3	GROUND	GROUND	BR1	BL1-U21E-1
4	LINE SYNC	(CLS)	BR1	BB1-U8B-6
5	GROUND	GROUND	BR1	BB1-U8B-6
6	INPUT PIXEL CLOCK	EARLY (CPE-BAR)	BR1	BB1-U8B-6
7	GROUND	GROUND	BR1	BB1-U8B-6
8	DAC PIXEL CLOCK	(CPD) (CHANGE TO BF1-10MHZ)	BR1	BB1
9	GROUND	GROUND	BR1	BB1
10	OUTPUT PIXEL CLOCK	(CPO)	BR1	BB1
11	GROUND	GROUND	BR1	C5-4
12	FIELD-1	(F1)	BR1	C5-5
13	GROUND	GROUND	BR1	C5-5
14	GROUND	GROUND	BR1	BR1-U11, U13
15	SEL0	ADC SELECT	LSB	DOA2
16	GROUND	GROUND	MSB	DOA1
17	SEL1	ADC SELECT	MSB	DOA1
18	GROUND	GROUND	BR1-U10-3	C6-18
19	DJ7	JOYSTICK COMMAND	MSB	BR1-U10-3
20	GROUND	GROUND	BR1-U10-3	C6-18
21	MO0R	BUFFER OUTPUT TO CRT	DACS (GREEN)	BB1-U19B-7 BR-U
22	GROUND	BUFFER OUTPUT TO CRT	DACS (GREEN)	BB1-U19B-7 BR-U
23	MO1R	BUFFER OUTPUT TO CRT	DACS (GREEN)	BB1-U19B-10BR-U
24	GROUND	BUFFER OUTPUT TO CRT	DACS (GREEN)	BB1-U19B-10BR-U
25	GROUND	GROUND		

26	MO2R	BUFFER	OUTPUT	TO	CRT	DACS	(GREEN)	BB1-U19B-12	BR-U
27	GROUND	GROUND							
28	MO3R	BUFFER	OUTPUT	TO	CRT	DACS	(RED)	BB1-U19B-15	BR-U
29	GROUND	GROUND							
30	MO4R	BUFFER	OUTPUT	TO	CRT	DACS	(RED)	BB1-U19C-2	BR-U
31	GROUND	GROUND							
32	MO5R	BUFFER	OUTPUT	TO	CRT	DACS	(BLUE)	BB1-U19C-5	BR-U
33	GROUND	GROUND							
34	MO6R	BUFFER	OUTPUT	TO	CRT	DACS	(BLUE)	BB1-U19C-7	BR-U
35	GROUND	GROUND							
36	MO7R	BUFFER	OUTPUT	TO	CRT	DACS	(SPARE)	BB1-U19C-10	
37	GROUND	GROUND							
38	DJ6	JOYSTICK	COMMAND					BR1-U10-5	C6-5
39	GROUND	GROUND							
40	DJ5	JOYSTICK	COMMAND					BR1-U10-7	C6-17
41	GROUND	GROUND							
42	DJ4	JOYSTICK	COMMAND					BR1-U10-9	C6-4
43	GROUND	GROUND							
44	DJ3	JOYSTICK	COMMAND					BR1-U10-11	C6-16
45	GROUND	GROUND							
46	DJ2	JOYSTICK	COMMAND					BR1-U10-13	C6-3
47	GROUND	GROUND							
48	DJ1	JOYSTICK	COMMAND					BR1-U13-13	C6-15
49	GROUND	GROUND							
50	DJ0	JOYSTICK	COMMAND					BR1-U13-11	C6-2

<u>PIN</u>	<u>SIGNAL</u>	<u>CABLE-V</u>	<u>BL1/COMPUTER</u>	<u>PORT-A</u>	<u>CONTROL</u>	<u>(C5)</u>
				<u>SOURCE</u>		<u>DESTINATION</u>
1	-12V					
2	DIA0	COMPUTER	DATA INPUT,	FRAME SYNC	C4-2	COMPUTER
3	DIA2	COMPUTER	DATA INPUT,	LINE SYNC	C4-4	COMPUTER
4	DIA4	COMPUTER	DATA INPUT,	FIELD 1-BAR	C4-12	COMPUTER
5	DIA6	COMPUTER	DATA INPUT			COMPUTER
6						
7	+5V					
8						
9	DOA0	COMPUTER	DATA OUTPUT,	TEST PULSE-1	COMPUTER	
10	DOA2	COMPUTER	DATA OUTPUT,	JOYSTICK SEL.-0	COMPUTER	C4-16
11	DOA4	COMPUTER	DATA OUTPUT,	TEST PULSE-16	COMPUTER	
12	DOA6	COMPUTER	DATA OUTPUT,	LOAD-BAR/RUN	COMPUTER	BL1-U18D-5
13	+12V					
14	GROUND	GROUND				
15	DIA1	COMPUTER	DATA INPUT			
16	DIA3	COMPUTER	DATA INPUT			
17	DIA5	COMPUTER	DATA INPUT			
18	DIA7	COMPUTER	DATA INPUT			
19	GROUND	GROUND				
20	GROUND	GROUND				
21						
22	DOA1	COMPUTER	DATA OUTPUT,	JOYSTICK SEL.-1	COMPUTER	C4-18
23	DOA3	COMPUTER	DATA OUTPUT,	UNBLANK	COMPUTER	C4-
24	DOA5	COMMAND SEQ.	LOAD OF IMAGE MEMORY		COMPUTER	BL1-U18E-12
25	DOA7	OUTPUT DATA STROBE			COMPUTER	BL1-U22B-2
26						

<u>CABLE-VI</u>	<u>BL1/COMPUTER</u>	<u>PORT-B</u>	<u>ADDRESS/DATA (C6)</u>	
<u>PIN</u>	<u>SIGNAL</u>	<u>DESCRIPTION</u>	<u>SOURCE</u>	<u>DESTINATION</u>
1	-12V			
2	DIB0	COMPUTER DATA INPUT,	JOYSTICK	C4-50 COMPUTER
3	DIB2	COMPUTER DATA INPUT,	JOYSTICK	C4-46 COMPUTER
4	DIB4	COMPUTER DATA INPUT,	JOYSTICK	C4-42 COMPUTER
5	DIB6	COMPUTER DATA INPUT,	JOYSTICK	C4-38 COMPUTER
6	+12V			
7	+5V			
8	DOB0	COMPUTER DATA OUTPUT	JOYSTICK	C2-2; U17C-3
9	DOB2	COMPUTER DATA OUTPUT	JOYSTICK	C2-6; U17C-6
10	DOB4	COMPUTER DATA OUTPUT	JOYSTICK	C2-10; U17C-13
11	DOB6	COMPUTER DATA OUTPUT	JOYSTICK	C2-14
12	+12V			
13	GROUND	GROUND	JOYSTICK	C4-48 COMPUTER
14	DIB1	COMPUTER DATA INPUT,	JOYSTICK	C4-44 COMPUTER
15	DIB3	COMPUTER DATA INPUT,	JOYSTICK	C4-40 COMPUTER
16	DIB5	COMPUTER DATA INPUT,	JOYSTICK	C4-20 COMPUTER
17	DIB7	COMPUTER DATA INPUT,	JOYSTICK	
18	GROUND	GROUND	JOYSTICK	
19	GROUND	GROUND	JOYSTICK	
20	GROUND	GROUND	JOYSTICK	
21	DOB1	COMPUTER DATA OUTPUT	COMPUTER	C2-4; U17C-4
22	DOB3	COMPUTER DATA OUTPUT	COMPUTER	C2-8; U17C-11
23	DOB5	COMPUTER DATA OUTPUT	COMPUTER	C2-12; U17C-14
24	DOB7	COMPUTER DATA OUTPUT	COMPUTER	C2-16
25				
26				

CABLE-VII		BL1/COMPUTER	PORT-C	REGISTER	SELECT	(C7)
PIN	SIGNAL	SIGNAL DESCRIPTION		SOURCE	DESTINATION	
1	-12V	COMPUTER DATA INPUT		COMPUTER		
2	DIC0	COMPUTER DATA INPUT		COMPUTER		
3	DIC2	COMPUTER DATA INPUT		COMPUTER		
4	DIC4	COMPUTER DATA INPUT		COMPUTER		
5	DIC6	COMPUTER DATA INPUT		COMPUTER		
6	+5V					
7						
8	DOC0	COMPUTER DATA OUTPUT		COMPUTER	BL1-U19B-1,	U20B-1
9	DOC2	COMPUTER DATA OUTPUT		COMPUTER	BL1-U19B-3,	U20B-3
10	DOC4	COMPUTER DATA OUTPUT		COMPUTER	BL1-U21D-5	
11	DOC6	COMPUTER DATA OUTPUT		COMPUTER		
12	+12V				BL1-U21D-2	
13						
14	GROUND	GROUND				
15	DIC1	COMPUTER DATA INPUT		COMPUTER		
16	DIC3	COMPUTER DATA INPUT		COMPUTER		
17	DIC5	COMPUTER DATA INPUT		COMPUTER		
18	DIC7	COMPUTER DATA INPUT		COMPUTER		
19	GROUND	GROUND				
20	GROUND	GROUND				
21						
22	DOC1	COMPUTER DATA OUTPUT		COMPUTER	BL1-U19B-2,	U20B-2
23	DOC3	COMPUTER DATA OUTPUT		COMPUTER	BL1-U19B-6,	U20B-5
24	DOC5	COMPUTER DATA OUTPUT		COMPUTER	BL1-U21D-4	
25	DOC7	COMPUTER DATA OUTPUT		COMPUTER	BL1-U21D-1	
26						

TABLE OF DIP LAYOUT ON BOARDS

BOARD-BM1,2    MEMORY BOARD

<u>POSITION</u>	<u>TYPE</u>	<u>PINS</u>	<u>SPARES</u>
U1A	58725	24	NONE
U1B	58725	24	NONE
U1C	58725	24	NONE
U1D	58725	24	NONE
U1E			
U2A	58725	24	NONE
U2B	58725	24	NONE
U2C	58725	24	NONE
U2D	58725	24	NONE
U2E			
U3A	58725	24	NONE
U3B	58725	24	NONE
U3C	58725	24	NONE
U3D	58725	24	NONE
U3E			
U4A	58725	24	NONE
U4B	58725	24	NONE
U4C	58725	24	NONE
U4D	58725	24	NONE
U4E			
U5A	58725	24	NONE
U5B	58725	24	NONE
U5C	58725	24	NONE
U5D	58725	24	NONE
U5E			
U6A	58725	24	NONE
U6B	58725	24	NONE
U6C	58725	24	NONE
U6D	58725	24	NONE
U6E			
U7A	58725	24	NONE
U7B	58725	24	NONE
U7C	58725	24	NONE
U7D	58725	24	NONE
U7E			

U8A	58725	24	NONE
U8B	58725	24	NONE
U8C	58725	24	NONE
U8D	58725	24	NONE
U8E			
U9A	58725	24	NONE
U9B	58725	24	NONE
U9C	58725	24	NONE
U9D	58725	24	NONE
U9E			
U10A	58725	24	NONE
U10B	58725	24	NONE
U10C	58725	24	NONE
U10D	58725	24	NONE
U10E			
U11A	58725	24	NONE
U11B	58725	24	NONE
U11C	58725	24	NONE
U11D	58725	24	NONE
U11E			
U12A	58725	24	NONE
U12B	58725	24	NONE
U12C	58725	24	NONE
U12D	58725	24	NONE
U12E			
U13A	58725	24	NONE
U13B	58725	24	NONE
U13C	58725	24	NONE
U13D	58725	24	NONE
U13E			
U14A	58725	24	NONE
U14B	58725	24	NONE
U14C	58725	24	NONE
U14D	58725	24	NONE
U14E			

U15A 58725 24 NONE  
U15B 58725 24 NONE  
U15C 58725 24 NONE  
U15D 58725 24 NONE  
U15E

U16A 58725 24 NONE  
U16B 58725 24 NONE  
U16C 58725 24 NONE  
U16D 58725 24 NONE  
U16E

U17A	8216	16	NONE	
U17B	8216	16	NONE	
U17C	8216	16	NONE	
U17D	8216	16	NONE	
U17E	LS00	14	NONE	
U18A	8216	16	NONE	
U18B	8216	16	NONE	
U18C	8216	16	NONE	
U18D	8216	16	NONE	
U18E	LS00	14	NONE	
U19A	8T97	16	NONE	(LS365 EQUIVALENT)
U19B	ALS138	16	NONE	
U19C	ALS138	16	NONE	
U19D	8T97	16	2/3	(LS365 EQUIVALENT)
U19E	LS138	16	NONE	
U20A				
U20B				
U20C				
U20D				
U20E				

BOARD-BB1 BUFFER BOARD

<u>POSITION</u>	<u>TYPE</u>	<u>PINS</u>	<u>SPARES</u>
U1A	LS175	16	12/10,11;13/15,14
U1B	LS02	14	5,6/4;8,9/10;11,12/13
U1C			
U1D			
U1E			
U1F			
U2A	ALS163	16	NONE
U2B	LS00	14	9,10/8;12,13/11
U2C			
U2D			
U2E			
U2F			
U3A	ALS163	16	NONE
U3B			
U3C	LS365	16	NONE
U3D	LS367	16	NONE
U3E	2149H	18	NONE
U3F			
U4A	ALS163	16	NONE
U4B			
U4C	LS365	16	NONE
U4D	LS367	16	NONE
U4E	2149H	18	NONE
U4F	ALS365	16	NONE
U5A	ALS161	16	NONE
U5B			
U5C	LS365	16	NONE
U5D	LS367	16	NONE
U5E	2149H	18	NONE
U5F	ALS365	16	NONE
U6A	ALS161	16	NONE
U6B	LS365	16	NONE
U6C	LS367	16	12/11;14/13
U6D	LS367	16	NONE
U6E	2149H	18	NONE
U6F	ALS367	16	12/11;14/13
U7A	ALS161	16	NONE
U7B	LS08	14	9,10/8;12,13/11
U7C	LS365	16	NONE
U7D	LS367	16	NONE
U7E	2149H	18	NONE
U7F			

U8A	ALS161	16	NONE
U8B	LS04	14	11/10;13/12
U8C	LS365	16	NONE
U8D	LS367	16	NONE
U8E	2149H	18	NONE
U8F	ALS365	16	NONE

U9A			
U9B			
U9C	LS365	16	NONE
U9D	LS367	16	NONE
U9E	2149H	18	NONE
U9F	ALS365	16	NONE

U10A			
U10B	LS365	16	NONE
U10C	LS367	16	12/11;14/13
U10D	LS367	16	NONE
U10E	2149H	18	NONE
U10F	ALS367	16	12/11;14/13

U11A			
U11B			
U11C	LS365	16	NONE
U11D	LS367	16	NONE
U11E	2149H	18	NONE
U11F			

U12A			
U12B	LS139	16	13,14,15/9,10,11,12
U12C	LS365	16	NONE
U12D	LS367	16	NONE
U12E	2149H	18	NONE
U12F	ALS365	16	NONE

U13A			
U13B	LS04	14	13/12
U13C	LS365	16	NONE
U13D	LS367	16	NONE
U13E	2149H	18	NONE
U13F	ALS365	16	NONE

U14A			
U14B	LS365	16	NONE
U14C	LS367	16	12/11;14/13
U14D	LS367	16	NONE
U14E	2149H	18	NONE
U14F	ALS367	16	12/11;14/13

U15A			
U15B			
U15C	LS365	16	NONE
U15D	LS367	16	NONE
U15E	2149H	18	NONE
U15F			
U16A			
U16B			
U16C	LS365	16	NONE
U16D	LS367	16	NONE
U16E	2149H	18	NONE
U16F	ALS365	16	NONE
U17A			
U17B			
U17C	LS365	16	NONE
U17D	LS367	16	NONE
U17E	2149H	18	NONE
U17F	ALS365	16	NONE
U18A			
U18B	LS365	16	NONE
U18C	LS367	16	12/11;14/13
U18D	LS367	16	NONE
U18E	2149H	18	NONE
U18F	ALS367	16	12/11;14/13
U19A	LS174	16	NONE
U19B	LS174	16	NONE
U19C	LS174	16	NONE
U19D	LS174	16	NONE
U19E			
U19F			
U20A	LS174	16	NONE
U20B	LS174	16	NONE
U20C	LS174	16	NONE
U20D	LS174	16	NONE
U20E			
U20F			
U21A	LS174	16	NONE
U21B	LS174	16	NONE
U21C	LS174	16	NONE
U21D	LS174	16	NONE
U21E	2149H	18	NONE
U21F			

U22A LS174 16 NONE  
U22B LS174 16 NONE  
U22C LS257 16 NONE  
U22D LS257 16 14,13/12;11,10/9  
U22E 2149H 18 NONE  
U22F

U23A  
U23B  
U23C LS367 16 NONE  
U23D LS367 16 NONE  
U23E 2149H 18 NONE  
U23F

U24A  
U24B  
U24C  
U24D  
U24E  
U24F

U25A  
U25B  
U25C  
U25D  
U25E  
U25F

U26A  
U26B  
U26C  
U26D  
U26E  
U26F

U27A  
U27B  
U27C  
U27D  
U27E  
U27F

U28A  
U28B  
U28C  
U28D  
U28E  
U28F

<u>POSITION</u>	<u>TYPE</u>	<u>PINS</u>	<u>SPARES</u>	<u>BOARD-BL1</u>	<u>LOGIC BOARD</u>
U1A					
U1B					
U1C					
U1D					
U1E					
U2A					
U2B					
U2C					
U2D					
U2E					
U3A		14			
U3B		14			
U3C		14			
U3D		14			
U3E		14			
U4A		14			
U4B		14			
U4C		14			
U4D		14			
U4E					
U5A					
U5B	LS174	16	NONE		
U5C	LS174	16	NONE		
U5D	LS174	16	NONE		
U5E	LS174	16	NONE		
U6A					
U6B	LS283	16	NONE		
U6C	LS283	16	NONE		
U6D	S283	16	NONE		
U6E	F283	16	NONE		
U7A					
U7B	LS174	16	NONE		
U7C	LS174	16	NONE		
U7D	LS174	16	NONE		
U7E	LS174	16	NONE		

U8A	LS04	14	11/10;13/12
U8B	LS174	16	NONE
U8C	LS174	16	NONE
U8D	AS174	16	NONE
U8E	AS174	16	NONE
U9A	AS21	14	NONE
U9B	LS174	16	NONE
U9C	LS174	16	NONE
U9D	AS174	16	NONE
U9E	AS174	16	NONE
U10A	AS21	14	NONE
U10B	LS257	16	NONE
U10C	LS257	16	NONE
U10D	ALS257	16	NONE
U10E	ALS257	16	NONE
U11A	LS02	14	2,3/1;5,6/4;8,9/10;11,12/13
U11B	LS257	16	NONE
U11C	LS257	16	NONE
U11D	ALS257	16	NONE
U11E	ALS257	16	NONE
U12A	AS20	14	NONE
U12B	LS257	16	NONE
U12C	LS257	16	NONE
U12D	ALS257	16	NONE
U12E	ALS257	16	NONE
U13A	AS20	14	NONE
U13B	LS283	16	NONE
U13C	LS283	16	NONE
U13D	S283	16	NONE
U13E	S283	16	NONE
U14A	AS04	14	9/8;11/10;13/12
U14B	LS283	16	NONE
U14C	LS283	16	NONE
U14D	S283	16	NONE
U14E	S283	16	NONE

U15A	LS08	14	9,10/8
U15B	LS283	16	NONE
U15C	LS283	16	NONE
U15D	F283	16	NONE
U15E	F283	16	NONE
U16A	ALS32	14	8,9/10;11,12/13
U16B		16	
U16C		16	
U16D	F283	16	NONE
U16E	F283	16	NONE
U17A	ALS00	14	NONE
U17B	LS174	16	NONE
U17C	LS174	16	NONE
U17D	LS174	16	NONE
U17E	LS174	16	NONE
U18A		14	
U18B	LS04	14	1/2;3/4;9/8;11/10;13/12;5/6
U18C	LS32	14	NONE
U18D	LS04	14	11/10
U18E	LS00	14	10/8
U19A	LS02	14	5,6/4; 8,9/10; 11,12/13
U19B	LS138	16	NONE
U19C	LS02	14	NONE
U19D	LS02	14	8,9/10
U19E		14	
U20A		14	
U20B	LS138	16	NONE
U20C	LS02	14	NONE
U20D	LS02	14	NONE
U20E	LS08	14	NONE
U21A			
U21B	LS174	16	6/7;11/10;13/12;14/15 (CPD CLOCK)
U21C	LS02	14	NONE
U21D	LS20	14	NONE
U21E	LS04	14	NONE

U22A	LS04	14	5/6;11/10
U22B	LS00	14	4,5/6;9,10/8;12,13/11
U22C	LS32	14	NONE
U22D		14	
U22E	ALS174	16	3/2;4/5
U23A	OSC	-	NONE
U23B			
U23C	LS174	16	6/7;13/12;14/15
U23D	ALS174	16	NONE
U23E			
U24A	OSC	-	NONE
U24B			
U24C			
U24D			
U24E			

MEMORY TABLE-A TO MEMORY TABLE-D

MEMORY TABLE-A

<u>RAM</u>	<u>PIN-1</u>	<u>PIN-2</u>	<u>PIN-3</u>	<u>PIN-4</u>	<u>PIN-5</u>	<u>PIN-6</u>
U1A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U2A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U3A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U4A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U5A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U6A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U7A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U8A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U9A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U10A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U11A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U12A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U13A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U14A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U15A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U16A	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U1B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U2B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U3B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U4B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U5B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U6B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U7B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U8B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U9B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U10B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U11B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U12B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U13B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U14B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U15B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U16B	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7

MEMORY TABLE-A (CONTINUED)

<u>RAM</u>	<u>PIN-7</u>	<u>PIN-8</u>	<u>PIN-9</u>	<u>PIN-10</u>	<u>PIN-11</u>	<u>PIN-12</u>
U1A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U2A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U3A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U4A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U5A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U6A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U7A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U8A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U9A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U10A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U11A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U12A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U13A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U14A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U15A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U16A	U19A-5	U19A-3	U17A-3	U17A-6	U17A-10	GROUND
U1B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U2B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U3B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U4B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U5B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U6B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U7B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U8B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U9B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U10B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U11B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U12B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U13B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U14B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U15B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND
U16B	U19A-5	U19A-3	U17B-3	U17B-6	U17B-10	GROUND

MEMORY TABLE-B

<u>RAM</u>	<u>PIN-1</u>	<u>PIN-2</u>	<u>PIN-3</u>	<u>PIN-4</u>	<u>PIN-5</u>	<u>PIN-6</u>
U1C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U2C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U3C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U4C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U5C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U6C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U7C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U8C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U9C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U10C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U11C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U12C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U13C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U14C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U15C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U16C	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U1D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U2D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U3D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U4D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U5D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U6D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U7D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U8D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U9D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U10D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U11D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U12D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U13D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U14D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U15D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7
U16D	U19D-7	U19D-5	U19A-13	U19A-11	U19A-9	U19A-7

MEMORY TABLE-B (CONTINUED)

<u>RAM</u>	<u>PIN-7</u>	<u>PIN-8</u>	<u>PIN-9</u>	<u>PIN-10</u>	<u>PIN-11</u>	<u>PIN-12</u>
U1C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U2C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U3C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U4C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U5C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U6C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U7C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U8C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U9C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U10C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U11C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U12C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U13C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U14C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U15C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U16C	U19A-5	U19A-3	U17C-3	U17C-6	U17C-10	GROUND
U1D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U2D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U3D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U4D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U5D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U6D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U7D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U8D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U9D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U10D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U11D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U12D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U13D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U14D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U15D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND
U16D	U19A-5	U19A-3	U17D-3	U17D-6	U17D-10	GROUND

MEMORY TABLE-C

<u>RAM</u>	<u>PIN-13</u>	<u>PIN-14</u>	<u>PIN-15</u>	<u>PIN-16</u>	<u>PIN-17</u>	<u>PIN-18</u>
U1A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U2A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U3A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U4A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U5A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U6A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U7A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U8A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-15
U9A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U10A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U11A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U12A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U13A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U14A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U15A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U16A	U17A-13	U18A-3	U18A-6	U18A-10	U18A-13	U19B-14
U1B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U2B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U3B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U4B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U5B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U6B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U7B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U8B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-13
U9B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12
U10B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12
U11B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12
U12B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12
U13B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12
U14B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12
U15B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12
U16B	U17B-13	U18B-3	U18B-6	U18B-10	U18B-13	U19B-12

MEMORY TABLE-C (CONTINUED)

<u>RAM</u>	<u>PIN-19</u>	<u>PIN-20</u>	<u>PIN-21</u>	<u>PIN-22</u>	<u>PIN-23</u>	<u>PIN-24</u>
U1A	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U2A	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U3A	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U4A	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U5A	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U6A	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U7A	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U8A	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC
U9A	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U10A	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U11A	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U12A	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U13A	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U14A	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U15A	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U16A	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC
U1B	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U2B	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U3B	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U4B	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U5B	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U6B	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U7B	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U8B	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC
U9B	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U10B	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U11B	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U12B	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U13B	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U14B	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U15B	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U16B	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC

MEMORY TABLE-D

<u>RAM</u>	<u>PIN-13</u>	<u>PIN-14</u>	<u>PIN-15</u>	<u>PIN-16</u>	<u>PIN-17</u>	<u>PIN-18</u>
U1C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U2C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U3C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U4C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U5C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U6C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U7C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U8C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-11
U9C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U10C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U11C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U12C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U13C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U14C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U15C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U16C	U17C-13	U18C-3	U18C-6	U18C-10	U18C-13	U19B-10
U1D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U2D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U3D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U4D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U5D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U6D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U7D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U8D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-9
U9D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7
U10D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7
U11D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7
U12D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7
U13D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7
U14D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7
U15D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7
U16D	U17D-13	U18D-3	U18D-6	U18D-10	U18D-13	U19B-7

MEMORY TABLE-D (CONTINUED)

<u>RAM</u>	<u>PIN-19</u>	<u>PIN-20</u>	<u>PIN-21</u>	<u>PIN-22</u>	<u>PIN-23</u>	<u>PIN-24</u>
U1C	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U2C	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U3C	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U4C	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U5C	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U6C	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U7C	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U8C	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC
U9C	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U10C	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U11C	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U12C	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U13C	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U14C	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U15C	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U16C	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC
U1D	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U2D	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U3D	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U4D	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U5D	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U6D	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U7D	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U8D	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC
U9D	U19D-13	U19C-15	U19E-15	U19D-11	U19D-9	VCC
U10D	U19D-13	U19C-14	U19E-14	U19D-11	U19D-9	VCC
U11D	U19D-13	U19C-13	U19E-13	U19D-11	U19D-9	VCC
U12D	U19D-13	U19C-12	U19E-12	U19D-11	U19D-9	VCC
U13D	U19D-13	U19C-11	U19E-11	U19D-11	U19D-9	VCC
U14D	U19D-13	U19C-10	U19E-10	U19D-11	U19D-9	VCC
U15D	U19D-13	U19C-9	U19E-9	U19D-11	U19D-9	VCC
U16D	U19D-13	U19C-7	U19E-7	U19D-11	U19D-9	VCC

BASIC PROGRAM LISTING

GRAPH.ASC

```

100      PRINT: PRINT: PRINT "FILE: GRAPH.ASC"
110      CLEAR
1940     R%=INP (236): S%=R% AND 1: IF S%=1 THEN 1940      'LOCKUP
ON VERT.SYNC=1
1980     R%=INP (236): S%=R% AND 16: IF S%=0 THEN 1940      'CHECK FIELD
1990     OUT 236,64      'INITIALIZE GRAPHICS GENERATOR
1992     R%=INP (236): S%=R% AND 1
1993     IF S%=0 THEN 1992      'LOCKUP ON VERT.SYNC=0
1994     OUT 236,0      'COMMAND LOAD, RUN-BAR
1995     OUT 238, 246: OUT 237, 0: OUT 236,128: OUT 236,0
'X-ROW MSH
1996     OUT 238, 247: OUT 237, 0: OUT 236,128: OUT 236,0
'X-ROW LSH
1997     OUT 238,242: OUT 237, 0: OUT 236,128: OUT 236,0
'X-PIXEL SLOPE MSH
1998     OUT 238,245: OUT 237, 0: OUT 236,128: OUT 236,0
'Y-PIXEL SLOPE MSH
1999     OUT 238,248: OUT 237, 0: OUT 236,128: OUT 236,0
'X-ROW SLOPE MSH
2000     OUT 238,251: OUT 237, 0: OUT 236,128: OUT 236,0
'Y-ROW SLOPE MSH
2001     OUT 238,244: OUT 237, 0: OUT 236,128: OUT 236,0
'Y-PIXEL SLOPE LSH
2002     OUT 238,240: OUT 237, 0: OUT 236,128: OUT 236,0
'X-ROW SLOPE LSH
2003     OUT 238,243: OUT 237, 255: OUT 236,128: OUT 236,0
'X-PIXEL SLOPE LSH
2004     OUT 238,241: OUT 237, 255: OUT 236,128: OUT 236,0
'Y-ROW SLOPE LSH
2005     OUT 236,80      'COMMAND RUN, LOAD-BAR ;PULSE-1 BRACKETING
COMPUTATION PERIOD
2006     R%=INP (236): S%=R% AND 1: IF S%=1 THEN 2006      'LOCKUP
ON VERT.SYNC=1
2007     R%=INP (236): S%=R% AND 16: IF S%=0 THEN 2006      'CHECK FIELD
2060     'ITERATIVE PROCESSING
2100     OUT 236,64
2140     'RESYNCHRONIZATION AND FIELD CONTROL PROCESSOR
2220     R%=INP (236): S%=R% AND 1
2300     IF S%=0 THEN 2220      'LOCKUP ON VERT.SYNC=0
3060     'INTERLACED SCAN CALCULATIONS
3100     'INPUT BYTE    128 064 032 016 008 004 002 001
3140           F2      F1      LS      FS
3180     OUT 236,0      'COMMAND LOAD, RUN-BAR
3220     R%=INP (236): S%=R% AND 16: IF S%=0 THEN 3540 ELSE 3260
'CHECK FIELD
3260           'FIELD-2
3300     'OUTPUT POSITION PARAMETERS
3340     OUT 238, 249: OUT 237, 0: OUT 236,128: OUT 236,0
'Y-ROW MSH
3380     OUT 238, 250: OUT 237, 4: OUT 236,128: OUT 236,0
'Y-ROW LSH
3500     GOTO 4140

```

3540 'FIELD-1  
3580 'OUTPUT POSITION PARAMETERS  
3620 OUT 238, 249: OUT 237, 0: OUT 236,128: OUT 236,0  
'Y-ROW MSH  
3660 OUT 238, 250: OUT 237, 0: OUT 236,128: OUT 236,0  
'Y-ROW LSH  
4140 OUT 236,80 'COMMAND RUN, LOAD-BAR ;PULSE-1 BRACKETING  
COMPUTATION PERIOD  
4220 GOTO 2060 'LOOP BACK FOR NEXT FIELD  
20000 END

BASIC PROGRAM LISTING

LD.ASC

```

50      PRINT "ACCESSED "LD" FILE TO LOAD IMAGE MEMORY:
REV.5/15/84 09:00"
55      INPUT "MURPHY (M) OR CAMILLE (C)":K200$
100     INT1%=0: D%=0: K8%=1: K9%=1
112     PRINT: PRINT "*****"
114     PRINT "      SELECT OPERATION"
116     PRINT "*****": PRINT
118     PRINT "RETURN TO OPERATING SYSTEM ..... 0"
120     PRINT "SELECT OVERLAY FOR LOADING INTO IMAGE MEMORY ... 1"
122     PRINT "SELECT IMAGE TO BE LOADED INTO IMAGE MEMORY"
124     PRINT "      CONCENTRIC SQUARE FRAMES ..... 2"
126     PRINT "      RECTANGLES AND LINES ..... 3"
128     PRINT "      SPIRALS ..... 4"
130     PRINT "      VIEWPORT COORDINATE SYMBOLS ..... 5"
132     PRINT "      PATTERN #6 ..... 6"
134     PRINT "      PATTERN #7 ..... 7"
136     PRINT "      SQUARE PATTERN ..... 8"
138     PRINT "      SQUARE FRAMES ..... 9"
140     PRINT "      PERIPHERAL SQUARES ..... 10"
141     PRINT "      PERIPHERAL TRIANGLES ..... 11"
142     PRINT "      HOUSE ..... 12"
151     INPUT "
NUMBER";A20%
152     IF A20%<13 THEN 155
153     PRINT "*****": PRINT "IMPROPER SELECTION":
PRINT "*****"
154     GOTO 112
155     IF A20%>0 THEN 158
156         SYSTEM
158     ON A20% GOSUB 170, 4400, 4530, 5500, 4500, 7500, 8500,
9000, 9040, 9180, 9280, 11070
159     GOTO 112
170     PRINT: PRINT "*****"
171     PRINT "      SELECT OVERLAY FOR LOADING INTO IMAGE MEMORY"
172     PRINT "*****": PRINT
173     PRINT "      RETURN TO MAIN MENU ..... 0"
174     PRINT "      SELECT RECTANGULAR IMAGE MEMORY PATTERN"
180     PRINT "      HORIZONTAL BARS"
200     PRINT "          3-2-2 WIDTH BARS, INTENSITY VARATIONS . 1"
220     PRINT "          1-1-1 WIDTH BARS, MAXIMUM INTENSITY .. 2"
240     PRINT "          LINEAR COUNT, ALL COLOR COMBINATIONS .. 8"
260     PRINT "          SOLID SINGLE COLORED IMAGES"
265     PRINT "          RECTANGLE ..... 3"
270     PRINT "          BACKGROUND ..... 4"
400     PRINT "          CHECKERBOARD"
420     PRINT "          4-COLORS ..... 6"
440     PRINT "          2-COLORS ..... 7"
442     PRINT "          VARIABLE SINGLE COLORS"
443     PRINT "          GREEN SAWTOOTH ..... 10"
460     PRINT "          CENTER ELEMENT"
480     PRINT "          9-PIXEL SQUARE ..... 11"
482     PRINT "          SELECT SLOPING LINE ..... 12"
484     INPUT "
NUMBER";A5%
486     IF A5%>0 THEN 502

```

```

500           RETURN
502   IF A5%<13 THEN 520
503   PRINT "*****": PRINT "IMPROPER SELECTION":
PRINT "*****"
504   GOTO 170
520   IF A5%=3 OR A5%=4 OR A5%=12 THEN 523 ELSE 535
523   PRINT "COLOR CODE"
524   PRINT "      BLACK ..... 0"
526   PRINT "      GREEN ..... 1 TO 7"
527   PRINT "      RED ..... 8, 16, 24"
528   PRINT "      BLUE ..... 32, 64, 96"
529   INPUT "                                SELECT COLOR
CODE SUM";INT1%
530   IF INT1%<128 THEN 535
531   PRINT:PRINT "*****"
532   PRINT "IMPROPER COLOR CODE; ENTER COLOR CODE AGAIN"
533   PRINT "*****":PRINT
534   GOTO 523
535   IF A5%=11 THEN 2040
540   IF A5%=6 OR A5%=7 GOTO 560 ELSE 570
560           INPUT "CHECKERBOARD RESOLUTION, PIXELS PER
SIDE";A6%
570   IF A5%=4 THEN 575 ELSE 580
575       A5%=3: XB%=0: YB%=0: XE%=511: YE%=511: GOTO 623
580   INPUT "      START PIXEL COORDINATE";XB%,YB%
620   INPUT "      STOP PIXEL COORDINATE";XE%,YE%
623   GOSUB 630
624   GOTO 170
630   *****
632   'SUBROUTINE TO OVERLAY A RECTANGLE
635       XB%=(XB%+1)*8: YB%=(YB%+1)*8
640       XE%=(XE%+1)*8: YE%=(YE%+1)*8
642'   IF XB%=>8 AND XB%<XE% AND XE%=>8 AND XB%=<512*8 AND
XE%=<512*8 THEN 643 ELSE 656
643'   IF YB%=>8 AND YB%<YE% AND YE%=>8 AND YB%=<512*8 AND
YE%=<512*8 THEN 660 ELSE 656
656'   PRINT:PRINT "*****"
657'   PRINT "IMPROPER PIXEL COORDINATES, ENTER PIXEL
COORDINATES AGAIN"
658'   PRINT "*****":PRINT
659'   GOTO 580
660'   PRINT: PRINT "*****"
665'   PRINT "      IMAGE MEMORY IS BEING LOADED"      'PRINT
"ROW", "COLOR INTENSITY"
666'   PRINT "*****": PRINT
667   IF A5%=12 THEN 4200
690   XPS%=256: YPS%=0: XRP%=XB%-8
695   GOSUB 3000
700   FOR OUTLP1%=YB% TO YE% STEP 8          'ROW LOOP
710   YRP%=OUTLP1%-8             'UPDATE TO NEXT ROW
711   XRP%=XB%-8
712   GOSUB 3000                 'LOAD IMP REGISTERS
948   OUT 236,32                'SET SEQUENTIAL LOAD COMMAND
1230  'DETERMINE INTENSITY (INT1%
1240  ON A5% GOTO

```

```

1260,1460,1720,1250,1250,1920,1980,1800,1250,1820
1250      PRINT "*****": PRINT "SELECT A DIFFERENT IMAGE":
PRINT "*****"
1260      IF D%<8 THEN 1340      'IMAGE PATTERN 1
1280      IF D%<32 THEN 1380    ' !-64-32-!-16-08-!-04-02-01-!
1300      IF D%<128 THEN 1420   ' ! BLUE ! RED ! GREEN !
1320      INT1%=0: D%=0: GOTO 1980
1340      INT1%=D% AND 7
1360          D%=D%+1: GOTO 1980
1380      INT1%=D% AND 24
1400          D%=D%+8: GOTO 1980
1420      INT1%=D% AND 96
1440          D%=D%+32: GOTO 1980
1460      A3%=FIX ((OUTLP1%-8)/8)           'IMAGE PATTERN 2
1480      GOSUB 1520
1500      GOTO 1980
1520      ***** SUBROUTINE, MAXIMUM COLOR *****
1540          A4%=A3% AND 3
1560          A7%=A4%+1
1580          ON A7% GOTO 1600,1620,1640,1660
1600          INT1%=0: GOTO 1680
1620          INT1%=7: GOTO 1680
1640          INT1%=24: GOTO 1680
1660          INT1%=96
1680      RETURN
1700  *****
1720          INT1%=INT1%: GOTO 1980           'PATTERN-3, SOLID
COLOR
1800          K2%=(OUTLP1%-8)/8: INT1%= K2% AND 127: GOTO 1980
'PATTERN 8, LINEAR COUNT
1820          INT1%=K8%: K10%=K9% AND 1
1840          IF K10%=0 THEN 1920
1860          K8%=K8%+1           'UPCOUNT, ADD
1880          IF K8%<8 THEN 1980
1900          K9%=K9%+1: K8%=6: GOTO 1980           'CHANGE
COUNT DIRECTION
1920          K8%=K8%-1           'DOWNCOUNT, SUBT
1940          IF K8%>0 THEN 1980
1960          K9%=K9%+1: K8%=2: GOTO 1980
'CHANGE COUNT DIRECTION
1980          SP1%=INIT1% AND 1: SP2%= INIT1% AND 3: SP3%=INT1% AND 7:
SP4%=INT1% AND 15: SP5%=INT1% AND 31: SP6%=INT1% AND 63:
SP7%=INT1% AND 127
1985          OUT 238, 252: OUT 237, SP1%: OUT 237, SP2%: OUT 237,
SP3%: OUT 237, SP4%: OUT 237, SP5%: OUT 237, SP6%: OUT 237, SP7%:
OUT 237, INT1% 'DATA TO LOAD IN IMAGE MEMORY
1991          ON A5% GOTO
1994,1994,1994,1994,1994,1992,1993,1994,1994,1994
1992          A3%=FIX((2*OUTLP1%+INLP1%)/A6%): GOSUB 1520: GOTO 1994
1993          A3%=FIX((2*OUTLP1%+2*INLP1%)/A6%): GOSUB 1520: GOTO 1994
1994          FOR INLP1%=XB% TO XE% STEP 8           'PIXEL LOOP
1995          OUT 236,160: OUT 236,32
1996          NEXT INLP1%
1997          OUT 237, SP7%: OUT 237, SP6%: OUT 237, SP5%: OUT 237,
SP4%: OUT 237, SP3%: OUT 237, SP2%: OUT 237, SP1%: OUT 237, 0

```

```

'DATA TO LOAD IN IMAGE MEMORY
1999    IF K200$="M" THEN
2000        A8%=INP (93): A8%=A8% AND 2: IF A8%=0 THEN 2009
'OPERATOR RESET
2001        A8%=INP (92): GOTO 2006
2002        A8%=INP (1): A8%=A8% AND 2: IF A8%=0 THEN 2009
'OPERATOR RESET
2003        A8%=INP (0)
2006            A9%=A8% XOR 155: IF A9%=0 THEN 100 'ESCAPE TO MENU
2007            A9%=A8% XOR 127: IF A9%=0 THEN 2008 ELSE 2009
'DELETE TO SYSTEM
2008        SYSTEM
2009        NEXT OUTLP1%
2010        PRINT CHR$(7); :PRINT "MEMORY LOAD COMPLETE"
2020        RETURN          'RETURN TO OVERLAY MENU
2040        'PATTERN 11
2060        INT1%=7: K17%=(256-3)*8: K18%=(256+3)*8
2080        FOR OUTLP1%=K17% TO K18% STEP 8
2100            OUT 238, 249          'Y-ROW MSH
2120            C%=FIX(OUTLP1%/64): OUT 237, C%: OUT 236,129: OUT 236,1
2140            OUT 238, 250          'Y-ROW LSH
2160            C%=OUTLP1% AND 63: OUT 237, C%: OUT 236,129: OUT 236,1
2180        FOR INLP1%=K17% TO K18% STEP 8          'PIXEL LOOP
2200            OUT 238, 246          'X-ROW MSH
2220            C%=FIX(INLP1%/64): OUT 237, C%: OUT 236,129: OUT 236,1
2240            OUT 238, 247          'X-ROW LSH
2260            C%=INLP1% AND 63: OUT 237, C%: OUT 236,129: OUT 236,1
2280            OUT 238, 252          'DATA TO LOAD IN IMAGE MEMORY
2300        OUT 237, INT1%: OUT 236,129: OUT 236,1
2320        NEXT INLP1%
2340        NEXT OUTLP1%
2360            OUTLP1%=(256-5)*8
2380            OUT 238, 249          'Y-ROW MSH
2400            C%=FIX(OUTLP1%/64): OUT 237, C%: OUT 236,129: OUT 236,1
2420            OUT 238, 250          'Y-ROW LSH
2440            C%=OUTLP1% AND 63: OUT 237, C%: OUT 236,129: OUT 236,1
2460            K17%=(256-6)*8: K18%=(256)*8
2480        FOR INLP1%=K17% TO K18% STEP 8          'PIXEL LOOP
2500            OUT 238, 246          'X-ROW MSH
2520            C%=FIX(INLP1%/64): OUT 237, C%: OUT 236,129: OUT 236,1
2540            OUT 238, 247          'X-ROW LSH
2560            C%=INLP1% AND 63: OUT 237, C%: OUT 236,129: OUT 236,1
2580            OUT 238, 252          'DATA TO LOAD IN IMAGE MEMORY
2600        OUT 237, INT1%: OUT 236,129: OUT 236,1
2620        NEXT INLP1%
2640        GOTO 2000
3000        ***** SUBROUTINE TO OUTPUT POSITION AND SLOPE PARAMETERI
3001        OUT 236,0          'DOA5 TURNED OFF TO DISABLE SEQUENCING
DURING LOADING OF REGISTERS
3002        'SLOPE SCALE FACTOR=256*PIXELS/STEP
3003        'POSITION SCALE FACTOR =8*PIXELS
3004        XPSM%=FIX(XPS%/64): XPSL%=XPS% AND 63
3005        OUT 238,242: OUT 237,XPSM%: OUT 236,128: OUT 236,0
'X-PIXEL SLOPE MSH
3006        OUT 238,243: OUT 237,XPSL%: OUT 236,128: OUT 236,0

```

```

'X-PIXEL SLOPE LSH
3007      YPSM%=FIX(YPS%/64):      YPSL%=YPS% AND 63
3008          OUT 238,245: OUT 237,YPSM%: OUT 236,128: OUT 236,0
'Y-PIXEL SLOPE MSH
3009          OUT 238,244: OUT 237,YPSL%: OUT 236,128: OUT 236,0
'Y-PIXEL SLOPE LSH
3010'     OUT 236, 0      'SUBROUTINE ENTRY POINT
3020      XRPM%=FIX(XRP%/64):      XRPL%=XRP% AND 63
3091      YRPM%=FIX(YRP%/64):      YRPL%=YRP% AND 63
'FORMAT POSITION OUTPUTS
3095          OUT 238,249: OUT 237,YRPM%: OUT 236,128: OUT 236,0
'Y-ROW MSH (Y-PIXEL MSH)
3096          OUT 238,250: OUT 237,YRPL%: OUT 236,128: OUT 236,0
'Y-ROW LSH (Y-PIXEL LSH)
3097          OUT 238,246: OUT 237,XRPM%: OUT 236,128: OUT 236,0
'X-ROW MSH (X-PIXEL MSH)
3098          OUT 238,247: OUT 237,XRPL%: OUT 236,128: OUT 236,0
'X-ROW LSH (X-PIXEL LSH)
3690      RETURN
4200      ****
4210      'DRAW A LINE
4220      DX=XE%-XB%: DY=YE%-YB%
4221      DTG=0
4222      IF DX=0 AND DY=0 THEN 4228
4224      IF ABS(DX)>ABS(DY) THEN 4227
4226      YPS%=(DY*256)/ABS(DY): XPS%=(DX*256)/ABS(DY):
DTG=ABS(DY): GOTO 4228
4227      YPS%=(DY*256)/ABS(DX): XPS%=(DX*256)/ABS(DX):
DTG=ABS(DX)
4228      XRP%=XB%-8: YRP%=YB%-8
4255      GOSUB 3000
4270      OUT 238, 252: OUT 237, INT1%           'DATA TO LOAD IN
IMAGE MEMORY
4273      OUT 236,32           'SET SEQUENTIAL LOAD COMMAND
4274      DTG=DTG+8
4276      IF DTG>8 THEN 4288
4287      OUT 236,160: OUT 236,32: GOTO 4305
4288      FOR INLP1=8 TO DTG STEP 8           'PIXEL LOOP
4290      OUT 236,160: OUT 236,32
4300      NEXT INLP1
4305      OUT 236,0           'RESET SEQUENTIAL LOAD COMMAND
4350      RETURN
4400      ****
4410      INIT1%=7: XB%=225: YB%=255: YPS%=0: XPS%=256
4420      FOR OUTLP1%=1 TO 10
4435      XRP%=XB%: YRP%=YB%
4437      XB%=XB%-1: YB%=YB%+1
4440      DX=2*OUTLP1%+1
4470      GOSUB 3000
4480      OUT 238, 252: OUT 237, INT1%           'DATA TO LOAD IN
IMAGE MEMORY
4484      OUT 236,32           'SET SEQUENTIAL LOAD COMMAND
4488      FOR INLP1=8 TO DX STEP 8           'PIXEL LOOP
4492      OUT 236,160: OUT 236,32
4493      NEXT INLP1

```

```

4495 OUT 236,0           'RESET SEQUENTIAL LOAD COMMAND
4496 NEXT OUTLP1%
4497 RETURN
4500 ****
4510 'BLACK BACKGROUND WITH COORDINATE SYMBOLS
4520 A5%=3: INT1%=0: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
4522 A5%=12: INT1%=3: XB%=0: YB%=0: XE%=0: YE%=511:
GOSUB 630
4523 A5%=12: INT1%=3: XB%=0: YB%=0: XE%=511: YE%=0:
GOSUB 630
4524 A5%=3: INT1%=3: XB%=252: YB%=252: XE%=258: YE%=258:
GOSUB 630
4525 A5%=3: INT1%=3: XB%=250: YB%=250: XE%=260: YE%=260:
GOSUB 630
4526 A5%=3: INT1%=3: XB%=0: YB%=0: XE%=10: YE%=10:
GOSUB 630
4527 RETURN
4530 ****
4531 'RECTANGLE AND LINE PATTERN
4540 A5%=3: INT1%=24: XB%=100: YB%=400: XE%=200: YE%=500:
GOSUB 630
4550 A5%=1:             XB%=400: YB%=100: XE%=500: YE%=200:
GOSUB 630
4555 A5%=10:            XB%=100: YB%=100: XE%=200: YE%=200:
GOSUB 630
4556 A5%=12: INT1%=24: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
4557 A5%=12: INT1%=24: XB%=0: YB%=511: XE%=511: YE%=0:
GOSUB 630
4561 RETURN
5500 ****
5510 'SPIRAL LINES
5535 A5%=12: INT1%=24: XB%=0: YB%=0: XE%=10000:
YE%=200: GOSUB 630
5540 A5%=12: INT1%=96: XB%=0: YB%=511: XE%=30000:
YE%=200: GOSUB 630
5561 RETURN
6500 ****
6520 A5%=3: INT1%=96: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
6530 A5%=3: INT1%=24: XB%=100: YB%=500: XE%=200: YE%=400:
GOSUB 630
6550 A5%=1:             XB%=500: YB%=100: XE%=400: YE%=200:
GOSUB 630
6560 A5%=10:            XB%=100: YB%=100: XE%=200: YE%=200:
GOSUB 630
6561 RETURN
7500 ****
7520 A5%=3: INT1%=96: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
7530 A5%=3: INT1%=24: XB%=100: YB%=500: XE%=200: YE%=400:
GOSUB 630
7535 A5%=12: INT1%=24: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630

```

```

7540    A5%=12: INT1%=24: XB%=0: YB%=511: XE%=511: YE%=0:
GOSUB 630
7550    A5%=1:           XB%=500: YB%=100: XE%=400: YE%=200:
GOSUB 630
7560    A5%=10:          XB%=100: YB%=100: XE%=200: YE%=200:
GOSUB 630
7561    RETURN
8500    ****
8520    A5%=3: INT1%=96: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
8530    A5%=3: INT1%=24: XB%=100: YB%=500: XE%=200: YE%=400:
GOSUB 630
8535    A5%=12: INT1%=24: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
8540    A5%=12: INT1%=24: XB%=0: YB%=511: XE%=511: YE%=0:
GOSUB 630
8550    A5%=1:           XB%=500: YB%=100: XE%=400: YE%=200:
GOSUB 630
8560    A5%=10:          XB%=100: YB%=100: XE%=200: YE%=200:
GOSUB 630
8561    RETURN
9000    ****
9001    A5%=3: INT1%=7:  XB%=128: YB%=128: XE%=256: YE%=256:
GOSUB 630
9010    A5%=3: INT1%=16: XB%=256: YB%=128: XE%=384: YE%=256:
GOSUB 630
9020    A5%=3: INT1%=96: XB%=256: YB%=256: XE%=384: YE%=384:
GOSUB 630
9030    A5%=3: INT1%=24: XB%=128: YB%=256: XE%=256: YE%=386:
GOSUB 630
9031    RETURN
9032    ****
9040    A5%=3: INT1%=32: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
9050    A5%=3: INT1%=7:  XB%=104: YB%=104: XE%=127: YE%=407:
GOSUB 630
9060    A5%=3: INT1%=7:  XB%=104: YB%=104: XE%=407: YE%=127:
GOSUB 630
9070    A5%=3: INT1%=7:  XB%=104: YB%=384: XE%=407: YE%=407:
GOSUB 630
9080    A5%=3: INT1%=7:  XB%=384: YB%=104: XE%=407: YE%=407:
GOSUB 630
9090    A5%=3: INT1%=24: XB%=150: YB%=150: XE%=173: YE%=361:
GOSUB 630
9100    A5%=3: INT1%=24: XB%=150: YB%=150: XE%=361: YE%=173:
GOSUB 630
9110    A5%=3: INT1%=24: XB%=150: YB%=338: XE%=361: YE%=361:
GOSUB 630
9120    A5%=3: INT1%=24: XB%=338: YB%=150: XE%=361: YE%=361:
GOSUB 630
9130    A5%=3: INT1%=96: XB%=196: YB%=196: XE%=219: YE%=315:
GOSUB 630
9140    A5%=3: INT1%=96: XB%=196: YB%=196: XE%=315: YE%=219:
GOSUB 630
9150    A5%=3: INT1%=96: XB%=196: YB%=292: XE%=315: YE%=315:

```

```

GOSUB 630
9160 A5%=3: INT1%=96: XB%=292: YB%=196: XE%=315: YE%=315:
GOSUB 630
9170 A5%=3: INT1%=7: XB%=242: YB%=242: XE%=269: YE%=269:
GOSUB 630
9171 RETURN
9172 ****
9180 A5%=3: INT1%=96: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
9190 A5%=1: XB%=96: YB%=96: XE%=159: YE%=159:
GOSUB 630
9200 A5%=1: XB%=352: YB%=352: XE%=415: YE%=415:
GOSUB 630
9210 A5%=10: XB%=96: YB%=352: XE%=159: YE%=415:
GOSUB 630
9220 A5%=10: XB%=352: YB%=96: XE%=415: YE%=159:
GOSUB 630
9230 A5%=3: INT1%=24: XB%=224: YB%=224: XE%=287: YE%=287:
GOSUB 630
9240 A5%=12: INT1%=7: XB%=159: YB%=159: XE%=224: YE%=224:
GOSUB 630
9250 A5%=12: INT1%=7: XB%=287: YB%=287: XE%=352: YE%=352:
GOSUB 630
9260 A5%=12: INT1%=7: XB%=159: YB%=352: XE%=224: YE%=287:
GOSUB 630
9270 A5%=12: INT1%=7: XB%=287: YB%=224: XE%=352: YE%=159:
GOSUB 630
9271 RETURN
9272 ****
9280 A5%=3: INT1%=96: XB%=0: YB%=0: XE%=511: YE%=511:
GOSUB 630
9300 A5%=12: INT1%=7: XB%=5: YB%=6: XE%=252: YE%=253:
GOSUB 630
9310 A5%=12: INT1%=7: XB%=259: YB%=260: XE%=506: YE%=507:
GOSUB 630
9320 A5%=12: INT1%=7: XB%=5: YB%=507: XE%=252: YE%=260:
GOSUB 630
9330 A5%=12: INT1%=7: XB%=259: YB%=253: XE%=506: YE%=6:
GOSUB 630
9340 A5%=12: INT1%=96: XB%=5: YB%=5: XE%=252: YE%=252:
GOSUB 630
9350 A5%=12: INT1%=96: XB%=259: YB%=259: XE%=506: YE%=506:
GOSUB 630
9360 A5%=12: INT1%=96: XB%=5: YB%=506: XE%=252: YE%=259:
GOSUB 630
9370 A5%=12: INT1%=96: XB%=259: YB%=252: XE%=506: YE%=5:
GOSUB 630
9380 A5%=12: INT1%=24: XB%=5: YB%=4: XE%=252: YE%=251:
GOSUB 630
9390 A5%=12: INT1%=24: XB%=259: YB%=258: XE%=506: YE%=505:
GOSUB 630
9400 A5%=12: INT1%=24: XB%=5: YB%=505: XE%=252: YE%=258:
GOSUB 630
9410 A5%=12: INT1%=24: XB%=259: YB%=251: XE%=506: YE%=4:
GOSUB 630

```

9420	A5%=3:	INT1%=7:	XB%=0:	YB%=0:	XE%=511:	YE%=0:
GOSUB	630					
9430	A5%=3:	INT1%=7:	XB%=0:	YB%=0:	XE%=0:	YE%=511:
GOSUB	630					
9440	A5%=3:	INT1%=7:	XB%=511:	YB%=0:	XE%=511:	YE%=511:
GOSUB	630					
9450	A5%=3:	INT1%=7:	XB%=0:	YB%=511:	XE%=511:	YE%=511:
GOSUB	630					
9460	A5%=3:	INT1%=96:	XB%=1:	YB%=1:	XE%=510:	YE%=1:
GOSUB	630					
9470	A5%=3:	INT1%=96:	XB%=1:	YB%=1:	XE%=1:	YE%=510:
GOSUB	630					
9480	A5%=3:	INT1%=96:	XB%=510:	YB%=1:	XE%=510:	YE%=510:
GOSUB	630					
9490	A5%=3:	INT1%=96:	XB%=1:	YB%=510:	XE%=510:	YE%=510:
GOSUB	630					
9500	A5%=3:	INT1%=24:	XB%=2:	YB%=2:	XE%=509:	YE%=2:
GOSUB	630					
9510	A5%=3:	INT1%=24:	XB%=2:	YB%=2:	XE%=2:	YE%=509:
GOSUB	630					
9520	A5%=3:	INT1%=24:	XB%=509:	YB%=2:	XE%=509:	YE%=509:
GOSUB	630					
9530	A5%=3:	INT1%=24:	XB%=2:	YB%=509:	XE%=509:	YE%=509:
GOSUB	630					
9540	A5%=3:	INT1%=7:	XB%=3:	YB%=3:	XE%=508:	YE%=3:
GOSUB	630					
9550	A5%=3:	INT1%=7:	XB%=3:	YB%=3:	XE%=3:	YE%=508:
GOSUB	630					
9560	A5%=3:	INT1%=7:	XB%=508:	YB%=3:	XE%=508:	YE%=508:
GOSUB	630					
9570	A5%=3:	INT1%=7:	XB%=3:	YB%=508:	XE%=508:	YE%=508:
GOSUB	630					
9580	A5%=3:	INT1%=96:	XB%=4:	YB%=4:	XE%=507:	YE%=4:
GOSUB	630					
9590	A5%=3:	INT1%=96:	XB%=4:	YB%=4:	XE%=4:	YE%=507:
GOSUB	630					
9600	A5%=3:	INT1%=96:	XB%=507:	YB%=4:	XE%=507:	YE%=507:
GOSUB	630					
9610	A5%=3:	INT1%=96:	XB%=4:	YB%=507:	XE%=507:	YE%=507:
GOSUB	630					
9620	A5%=3:	INT1%=24:	XB%=5:	YB%=5:	XE%=506:	YE%=5:
GOSUB	630					
9630	A5%=3:	INT1%=24:	XB%=5:	YB%=5:	XE%=5:	YE%=506:
GOSUB	630					
9640	A5%=3:	INT1%=24:	XB%=506:	YB%=5:	XE%=506:	YE%=506:
GOSUB	630					
9650	A5%=3:	INT1%=24:	XB%=5:	YB%=506:	XE%=506:	YE%=506:
GOSUB	630					
9660	A5%=12:	INT1%=7:	XB%=192:	YB%=128:	XE%=319:	YE%=128:
GOSUB	630					
9670	A5%=12:	INT1%=96:	XB%=193:	YB%=129:	XE%=318:	YE%=129:
GOSUB	630					
9680	A5%=12:	INT1%=24:	XB%=194:	YB%=130:	XE%=317:	YE%=130:
GOSUB	630					
9690	A5%=12:	INT1%=7:	XB%=200:	YB%=136:	XE%=311:	YE%=136:

GOSUB 630  
 9700 A5%=12: INT1%=96: XB%=201: YB%=137: XE%=310: YE%=137:  
 GOSUB 630  
 9710 A5%=12: INT1%=24: XB%=202: YB%=138: XE%=309: YE%=138:  
 GOSUB 630  
 9720 A5%=12: INT1%=7: XB%=208: YB%=144: XE%=303: YE%=144:  
 GOSUB 630  
 9730 A5%=12: INT1%=96: XB%=209: YB%=145: XE%=302: YE%=145:  
 GOSUB 630  
 9740 A5%=12: INT1%=24: XB%=210: YB%=146: XE%=301: YE%=146:  
 GOSUB 630  
 9750 A5%=12: INT1%=7: XB%=216: YB%=152: XE%=295: YE%=152:  
 GOSUB 630  
 9760 A5%=12: INT1%=96: XB%=217: YB%=153: XE%=294: YE%=153:  
 GOSUB 630  
 9770 A5%=12: INT1%=24: XB%=218: YB%=154: XE%=293: YE%=154:  
 GOSUB 630  
 9780 A5%=12: INT1%=7: XB%=224: YB%=160: XE%=287: YE%=160:  
 GOSUB 630  
 9790 A5%=12: INT1%=96: XB%=225: YB%=161: XE%=286: YE%=161:  
 GOSUB 630  
 9800 A5%=12: INT1%=24: XB%=226: YB%=162: XE%=285: YE%=162:  
 GOSUB 630  
 9810 A5%=12: INT1%=7: XB%=232: YB%=168: XE%=279: YE%=168:  
 GOSUB 630  
 9820 A5%=12: INT1%=96: XB%=233: YB%=169: XE%=278: YE%=169:  
 GOSUB 630  
 9830 A5%=12: INT1%=24: XB%=234: YB%=170: XE%=277: YE%=170:  
 GOSUB 630  
 9840 A5%=12: INT1%=7: XB%=240: YB%=176: XE%=271: YE%=176:  
 GOSUB 630  
 9850 A5%=12: INT1%=96: XB%=241: YB%=177: XE%=270: YE%=177:  
 GOSUB 630  
 9860 A5%=12: INT1%=24: XB%=242: YB%=178: XE%=269: YE%=178:  
 GOSUB 630  
 9870 A5%=12: INT1%=7: XB%=248: YB%=184: XE%=263: YE%=184:  
 GOSUB 630  
 9890 A5%=12: INT1%=96: XB%=249: YB%=185: XE%=262: YE%=185:  
 GOSUB 630  
 9900 A5%=12: INT1%=24: XB%=250: YB%=186: XE%=261: YE%=186:  
 GOSUB 630  
 9910 A5%=3: INT1%=7: XB%=255: YB%=192: XE%=256: YE%=192:  
 GOSUB 630  
 9920 A5%=3: INT1%=7: XB%=255: YB%=420: XE%=256: YE%=420:  
 GOSUB 630  
 9930 A5%=12: INT1%=7: XB%=250: YB%=426: XE%=261: YE%=426:  
 GOSUB 630  
 9940 A5%=12: INT1%=96: XB%=249: YB%=427: XE%=262: YE%=427:  
 GOSUB 630  
 9950 A5%=12: INT1%=24: XB%=248: YB%=428: XE%=263: YE%=428:  
 GOSUB 630  
 9960 A5%=12: INT1%=7: XB%=242: YB%=434: XE%=269: YE%=434:  
 GOSUB 630  
 9970 A5%=12: INT1%=96: XB%=241: YB%=435: XE%=270: YE%=435:  
 GOSUB 630

```

9980    A5%=12: INT1%=24: XB%=240: YB%=436: XE%=271: YE%=436:
GOSUB 630
9990    A5%=12: INT1%=7:  XB%=234:  YB%=442:  XE%=277:  YE%=442:
GOSUB 630
10000   A5%=12: INT1%=96: XB%=233: YB%=443: XE%=278: YE%=443:
GOSUB 630
10010   A5%=12: INT1%=24: XB%=232: YB%=444: XE%=279: YE%=444:
GOSUB 630
10020   A5%=12: INT1%=7:  XB%=226:  YB%=450:  XE%=285:  YE%=450:
GOSUB 630
10030   A5%=12: INT1%=96: XB%=225: YB%=451: XE%=286: YE%=451:
GOSUB 630
10040   A5%=12: INT1%=24: XB%=224: YB%=452: XE%=287: YE%=452:
GOSUB 630
10050   A5%=12: INT1%=7:  XB%=218:  YB%=458:  XE%=293:  YE%=458:
GOSUB 630
10060   A5%=12: INT1%=96: XB%=217: YB%=459: XE%=294: YE%=459:
GOSUB 630
10070   A5%=12: INT1%=24: XB%=216: YB%=460: XE%=295: YE%=460:
GOSUB 630
10080   A5%=12: INT1%=7:  XB%=210:  YB%=466:  XE%=301:  YE%=466:
GOSUB 630
10090   A5%=12: INT1%=96: XB%=209: YB%=467: XE%=302: YE%=467:
GOSUB 630
11000   A5%=12: INT1%=24: XB%=208: YB%=468: XE%=303: YE%=468:
GOSUB 630
11010   A5%=12: INT1%=7:  XB%=202:  YB%=474:  XE%=309:  YE%=474:
GOSUB 630
11020   A5%=12: INT1%=96: XB%=201: YB%=475: XE%=310: YE%=475:
GOSUB 630
11030   A5%=12: INT1%=24: XB%=200: YB%=476: XE%=311: YE%=476:
GOSUB 630
11040   A5%=12: INT1%=7:  XB%=194:  YB%=482:  XE%=317:  YE%=482:
GOSUB 630
11050   A5%=12: INT1%=96: XB%=193: YB%=483: XE%=318: YE%=483:
GOSUB 630
11059   A5%=12: INT1%=24: XB%=192: YB%=484: XE%=319: YE%=484:
GOSUB 630
11060   A5%=3:  INT1%=24: XB%=248: YB%=248: XE%=263: YE%=263:
GOSUB 630
11061   RETURN
11062   ****
11070   A5%=3:  INT1%=7:  XB%=0:   YB%=0:   XE%=511:  YE%=256:
GOSUB 630
11080   A5%=3:  INT1%=96: XB%=0:  YB%=256:  XE%=511:  YE%=511:
GOSUB 630
11090   A5%=3:  INT1%=24: XB%=256: YB%=128: XE%=352: YE%=256:
GOSUB 630
12000   A5%=3:  INT1%=16: XB%=352: YB%=128: XE%=432: YE%=256:
GOSUB 630
12010   A5%=12: INT1%=0:  XB%=256: YB%=256: XE%=296: YE%=296:
GOSUB 630
12020   A5%=12: INT1%=0:  XB%=256: YB%=255: XE%=296: YE%=295:
GOSUB 630
12030   A5%=12: INT1%=0:  XB%=256: YB%=257: XE%=296: YE%=297:

```

GOSUB 630  
 12040 A5%=12: INT1%=0: XB%=296: YB%=296: XE%=392: YE%=296:  
 GOSUB 630  
 12050 A5%=12: INT1%=0: XB%=296: YB%=295: XE%=392: YE%=295:  
 GOSUB 630  
 12060 A5%=12: INT1%=0: XB%=296: YB%=397: XE%=392: YE%=297:  
 GOSUB 630  
 12070 A5%=12: INT1%=0: XB%=392: YB%=296: XE%=352: YE%=256:  
 GOSUB 630  
 12080 A5%=12: INT1%=0: XB%=392: YB%=295: XE%=352: YE%=255:  
 GOSUB 630  
 12090 A5%=12: INT1%=0: XB%=392: YB%=297: XE%=352: YE%=257:  
 GOSUB 630  
 13000 A5%=12: INT1%=0: XB%=256: YB%=256: XE%=352: YE%=256:  
 GOSUB 630  
 13010 A5%=12: INT1%=0: XB%=256: YB%=255: XE%=352: YE%=255:  
 GOSUB 630  
 13020 A5%=12: INT1%=0: XB%=256: YB%=257: XE%=352: YE%=257:  
 GOSUB 630  
 13030 A5%=12: INT1%=0: XB%=256: YB%=128: XE%=256: YE%=256:  
 GOSUB 630  
 13040 A5%=12: INT1%=0: XB%=256: YB%=127: XE%=256: YE%=255:  
 GOSUB 630  
 13050 A5%=12: INT1%=0: XB%=256: YB%=129: XE%=256: YE%=257:  
 GOSUB 630  
 13060 A5%=12: INT1%=0: XB%=256: YB%=128: XE%=432: YE%=128:  
 GOSUB 630  
 13070 A5%=12: INT1%=0: XB%=256: YB%=127: XE%=432: YE%=127:  
 GOSUB 630  
 13080 A5%=12: INT1%=0: XB%=256: YB%=129: XE%=432: YE%=129:  
 GOSUB 630  
 13090 A5%=12: INT1%=0: XB%=352: YB%=128: XE%=352: YE%=256:  
 GOSUB 630  
 14000 A5%=12: INT1%=0: XB%=352: YB%=127: XE%=352: YE%=255:  
 GOSUB 630  
 14010 A5%=12: INT1%=0: XB%=352: YB%=129: XE%=352: YE%=257:  
 GOSUB 630  
 14020 A5%=12: INT1%=0: XB%=392: YB%=296: XE%=432: YE%=256:  
 GOSUB 630  
 14030 A5%=12: INT1%=0: XB%=392: YB%=295: XE%=432: YE%=255:  
 GOSUB 630  
 14040 A5%=12: INT1%=0: XB%=392: YB%=297: XE%=432: YE%=257:  
 GOSUB 630  
 14050 A5%=12: INT1%=0: XB%=432: YB%=256: XE%=432: YE%=128:  
 GOSUB 630  
 14060 A5%=12: INT1%=0: XB%=432: YB%=255: XE%=432: YE%=127:  
 GOSUB 630  
 14070 A5%=12: INT1%=0: XB%=432: YB%=257: XE%=432: YE%=129:  
 GOSUB 630  
 14080 A5%=12: INT1%=127: XB%=12: YB%=480: XE%=12: YE%=496:  
 GOSUB 630  
 14090 A5%=12: INT1%=127: XB%=12: YB%=496: XE%=16: YE%=500:  
 GOSUB 630  
 15000 A5%=12: INT1%=127: XB%=16: YB%=500: XE%=32: YE%=500:  
 GOSUB 630

15010 A5%=12: INT1%=127: XB%=32: YB%=500: XE%=36: YE%=496:  
GOSUB 630  
15020 A5%=12: INT1%=127: XB%=36: YB%=496: XE%=36: YE%=480:  
GOSUB 630  
15030 A5%=12: INT1%=127: XB%=36: YB%=480: XE%=32: YE%=476:  
GOSUB 630  
15040 A5%=12: INT1%=127: XB%=32: YB%=476: XE%=16: YE%=476:  
GOSUB 630  
15050 A5%=12: INT1%=127: XB%=16: YB%=476: XE%=12: YE%=480:  
GOSUB 630  
15060 A5%=12: INT1%=127: XB%=6: YB%=476: XE%=12: YE%=480:  
GOSUB 630  
15070 A5%=12: INT1%=127: XB%=6: YB%=500: XE%=12: YE%=496:  
GOSUB 630  
15080 A5%=12: INT1%=127: XB%=12: YB%=506: XE%=16: YE%=500:  
GOSUB 630  
15090 A5%=12: INT1%=127: XB%=36: YB%=506: XE%=32: YE%=500:  
GOSUB 630  
16000 A5%=12: INT1%=127: XB%=42: YB%=500: XE%=36: YE%=496:  
GOSUB 630  
16010 A5%=12: INT1%=127: XB%=36: YB%=470: XE%=32: YE%=476:  
GOSUB 630  
16020 A5%=12: INT1%=127: XB%=12: YB%=470: XE%=16: YE%=476:  
GOSUB 630  
16030 A5%=12: INT1%=127: XB%=42: YB%=476: XE%=36: YE%=480:  
GOSUB 630  
16031 RETURN  
16032 \*\*\*\*\*  
40000 END

BASIC PROGRAM LISTING

FTR.ASC

```

10      LPRINT "FTR2.ASC"          '8/15/84           REV. 8/22/4    09:30"
20      'PREFILTER THE DATABASE IMAGE
30          OPEN "R", #1, "PRESENT.BIN", 128
40          FIELD 1,128 AS IAS
50      'SET UP INTEGETER CONSTANTS FOR SPEED
60      K1=1/256
70      K2=1/8
80      K3=1/32
90      RC1=0      'INITIALIZE TO ZERO
100     F5%=66        'INPUT "NUMBER OF LINES TO BE
110     FILTERED"; F5%
120     INPUT "ENABLE SMOOTHING PRINTOUTS; 'Y' OR 'N'" ; A15$
130     INPUT "ENABLE STORING PRINTOUTS; 'Y' OR 'N'" ; A16$
140     PRINT "SELECT FIXED WEIGHTS"
150     INPUT "SELECT STANDARD WEIGHTS; 'Y' OR 'N'" ; A1$
160     IF A1$ = "N" THEN 680
160     PRINT: PRINT
170     PRINT "SELECT", " 1 ", " 2 ", " 3 ": PRINT
180     PRINT , "2 4 2", "4 6 4", "8 8 8"
190     PRINT , "4 8 4", "6 8 6", "8 8 8"
200     PRINT , "2 4 2", "4 6 4", "8 8 8"
210     A1%=3      'PRINT: INPUT "SELECT WEIGHTS '1', '2', OR '3'" ; A1%
220     ON A1% GOTO 260, 290, 320
230     PRINT "*****"
240     PRINT "IMPROPER WEIGHT SELECTION, MAKE ANOTHER SELECTION"
250     PRINT "*****": PRINT: PRINT: PRINT:
GOTO 130
260     W1% = 2: W2% = 4: W3% = 2                  'WEIGHT KERNEL 1
270     W4% = 4: W5% = 8: W6% = 4
280     W7% = 2: W8% = 4: W9% = 2: GOTO 390
290     W1% = 4: W2% = 6: W3% = 4                  'WEIGHT KERNEL 2
300     W4% = 6: W5% = 8: W6% = 6
310     W7% = 4: W8% = 6: W9% = 4: GOTO 390
320     W1% = 8: W2% = 8: W3% = 8                  'WEIGHT KERNEL 3
330     W4% = 8: W5% = 8: W6% = 8
340     W7% = 8: W8% = 8: W9% = 8: GOTO 390
350     PRINT: PRINT "INPUT FIXED WEIGHTS"
360     INPUT "INPUT WEIGHTS; W1, W2, W3"; W1%, W2%, W3%
370     INPUT "INPUT WEIGHTS; W4, W5, W6"; W4%, W5%, W6%
380     INPUT "INPUT WEIGHTS; W7, W8, W9"; W7%, W8%, W9%
390     WTS1% = W1% + W2% + W3% + W4% + W5% + W6% + W7% + W8% + W9%      'WEIGHT
SCALE FACTOR
400     WTS2 = 1/WTS1%                                'RECIPROCAL
WEIGHT SCALE FACTOR
410     PRINT: PRINT , " WEIGHTS SELECTED": PRINT
420     PRINT , "W1="; W1%, "W2="; W2%, "W3="; W3%
430     PRINT , "W4="; W4%, "W5="; W5%, "W6="; W6%
440     PRINT , "W7="; W7%, "W8="; W8%, "W9="; W9%, "WTS1="; WTS1%: PRINT
450     WRR2% = 0: WRG2% = 0: WRB2% = 0: S1% = 0: S2% = 0
460     FOR KRLP1% = 1 TO F5%  'KERNEL ROW LOOP
465     IF KRLP1% > 1 THEN TST$ = "Y" ELSE TST$ = "N"
470     LPRINT "START KERNEL ROW="; KRLP1%
480     FOR KPLP1% = 1 TO 512  'KERNEL PIXEL LOOP
490     GOSUB 1820      'TERMINAL INPUT
500     XB% = KPLP1%: YB% = KRLP1%

```

```

510 XE% = XB% + 2: YE% = YB% + 2
520 X = (XB%/128) 'FIND PIXEL COORDINATE FOR RECORD LOCATION
530 BDC = XB%: IF BDC <=128 THEN 570
540 BDC = ABS(BDC-128) 'FIND BEGINNING DATA COUNT IN RECORD
550 'IF TST$ = "Y" THEN PRINT "PIXEL LOOP ";KPLP1%
560 IF BDC>128 THEN GOTO 540
570 FOR OUTLP1% = YB% TO YE% 'LINE LOOP
580 OUT 236,1
590 GOSUB 1820
600 GOSUB 2180 'DEFINE RECORD NO.
610 'IF TST$ = "Y" THEN PRINT "OUTLP1% ";OUTLP1%
620 'IF TST$ = "Y" THEN PRINT "RC ";RC
630 IF (RC >4) AND (RC < 261) THEN 640 ELSE 710
640 RC = RC-4 'SUBTRACT 4 SO LINES GO FROM 1-64
650 IF TOGL$="T" THEN 660 ELSE 860
660 CLOSE #1
670 OPEN "R",#1,"PRESENT.BIN",128 'REOPEN PRESENT FILE IF ANOTHER
680 FIELD 1,128 AS IA$ 'FILE WAS USED
690 TOGL$="F"
700 GOTO 860
710 IF RC < 5 THEN 720 ELSE 790
720 'GET RECORD FROM PREVIOUS FILE
730 CLOSE #1
740 OPEN "R",#1,"PRIOR.BIN",128
750 FIELD 1,128 AS IA$
760 TOGL$ = "T" 'FILE MUST BE OPEN FOR INNER LOOP
770 RC=252+RC 'SET RC TO LAST LINE IN PRIOR FILE
780 GOTO 860 'GET RECORD FROM FILE
790 'ELSE
800 'GET RECORD FROM NEXT FILE
810 CLOSE #1
820 OPEN "R",#1,"NEXT.BIN",128
830 FIELD 1,128 AS IA$
840 TOGL$ = "T" 'FILE MUST BE OPEN FOR INNER LOOP
850 RC=RC-260 'SET RC TO FIRST LINE IN NEXT ;FILE
860 GET #1,RC
870 FOR INLP1% = XB% TO XE% STEP 2 'PIXEL LOOP
880 OUT 236,1: OUT 236,0
890 ' CHECK FOR EVEN BYTE STARTING ADDRESS
900 IF A15$="Y" THEN GOSUB 1970
910 IF (INLP1% = XB% ) AND ((XB% MOD 2) = 0) THEN DC=BDC-1
920 IF (INLP1% = XB% ) AND ((XB% MOD 2) <> 0) THEN DC = BDC
930 A$=MID$(IA$,DC,2) :' GET 2 BYTES
940 'X IF CVI(A$) = 0 THEN K%=0: GOTO 1220
950 K% = CVI(A$)
960 PIXA% = K% AND 255 : ' GET FIRST BYTE
970 PIXB% = K% * K1 : 'GET SECOND BYTE
975 'IF TST$ = "Y" THEN PRINT "PIXA%, PIXB% ";PIXA%,PIXB%
980 'X OUT 236,16: OUT 236,0
990 'UNPACK RED, GREEN, AND BLUE INTENSITIES
1000 IF S1%=0 THEN 1010 ELSE 1020
1010 RDG2%=PIXB% AND 7: RDR2%=(PIXB% AND 24)*K2: RDB2%=(PIXB%
AND 96)*K3
1020 RDG1%=PIXA% AND 7: RDR1%=(PIXA% AND 24)*K2: RDB1%=(PIXA%
AND 96)*K3

```

```

1030 'X OUT 236,16: OUT 236,0
1040     S3%=OUTLP1%-YB%+1: ON S3% GOTO 1060, 1070, 1080
1050         LPRINT "ERROR AT LINE 1610
1060             WS1%=W1%: WS2%=W2%: WS3%=W3%: GOTO 1090
1070             WS1%=W4%: WS2%=W5%: WS3%=W6%: GOTO 1090
1080             WS1%=W7%: WS2%=W8%: WS3%=W9%
1090 'X OUT 236,16: OUT 236,0
1100     IF S1%=0 THEN 1140
1110     WRR2% = RDR1%*WS3% + WRR2%
1120     WRG2% = RDG1%*WS3% + WRG2%
1130     WRB2% = RDB1%*WS3% + WRB2%: GOTO 1170
1140     WRR2% = RDR1%*WS1% + RDR2%*WS2% + WRR2%
1150     WRG2% = RDG1%*WS1% + RDG2%*WS2% + WRG2%
1160     WRB2% = RDB1%*WS1% + RDB2%*WS2% + WRB2%: S1%=1
1170 'X OUT 236,16: OUT 236,0
1180     DC=DC+2: IF DC>=128 THEN 1190 ELSE 1200 'UPDATE DATA COUNT
1190     DC=1: RC=RC+1: GET #1,RC           'CONDITIONALLY UPDATE
RECORD COUNT
1200     NEXT INLP1%
1210     S1%=0
1220     NEXT OUTLP1%
1230     OUT 236,16
1240     IF S2%>0 THEN 1280
1250     S2%=1
1260     WRR1%=WRR2%*WTS2: WRG1%=WRG2%*WTS2: WRB1%=WRB2%*WTS2
1270     IF A15$="Y" THEN GOSUB 1270: GOTO 1310
1280     WRR2%=WRR2%*WTS2: WRG2%=WRG2%*WTS2: WRB2%=WRB2%*WTS2
1290     IF A15$="Y" THEN GOSUB 2000
1300     S2%=0: GOSUB 1410                 'STORE
1310     WRR2%=0: WRG2%=0: WRB2%=0
1320     OUT 236,0
1330     NEXT KPLP1%
1340     NEXT KRLP1%
1350     CLOSE #1
1360     OPEN "R",#2,"FILTERED.BIN",128
1370     FIELD 2,128 AS IB$
1380     PUT #2,RC1
1390     CLOSE
1400     SYSTEM
1410
-----
1420     '
1430     :      CODE TO PUT NEW BYTES TO SPECIFIED WRITE FILE
1440     :      THE NEW BYTES SHOULD BE IN THE WRITE VARIABLES
1450     :
1460     :      CLOSE READ FILE AND OPEN WRITE FILE
1470     :
1480     :
1490     :      CALCULATIONS FOR EXACT PIXEL LOCATION
1500     XX=XB%: YY=YB%
1510     X = (XX/128) : 'FIND PIXEL COORDINATE FOR RECORD LOCATION
1520     BDC = XX: IF BDC <=128 THEN 1550          'MODULO 128
1530     BDC = ABS(BDC-128)           : 'FIND BEGINNING DATA COUNT IN RECORD
1540     IF BDC>128 THEN GOTO 1530
1550     DC2 =BDC

```

```

1560 OUTLP1%=YY
1570 GOSUB 2180      'GET RECORD NO.
1580 'RC0=RC
1590 'IF TSTS = "Y" THEN PRINT "RC1, RC0 ";RC1,RC0
1600 'IF RC1 = RC0 THEN 1710
1610 CLOSE #1
1620 OPEN "R",#2,"FILTERED.BIN",128
1630 FIELD 2,128 AS IB$
1640 'IF RC1=0 THEN 1660
1645   IB$=IB2$
1650   'PUT #2,RC1: 'PRINT "EXECUTED PUT ";CVI(IB$)
1660   GET #2,RC
1665 'IB2$=IB$
1670 'RC1 = RC0
1680 'CLOSE #2
1690 'OPEN "R",#1,"PRESENT.BIN",128
1700 'FIELD 1,128 AS IA$
1710   ' PACK RED, GREEN, BLUE COLORS INTO THEIR RESPECTIVE BYTES
1720     WR1% = WRG1% OR 8 * WRR1%
1730     WR1% = WR1% OR 32 * WRB1%
1740     WR2% = WRG2% OR 8 * WRR2%
1750     WR2% = WR2% OR 32 * WRB2%
1760 N3 = (WR2%*256) OR WR1%          ' PACK 2 BYTES INTO 1 WORD
1770 A$ = MKI$(N3)
1780 'PRINT "N3 ";N3
1790 MID$(IB$,DC2,2)=A$
1795 'PRINT "A$ ";CVI(A$)
1796 PUT #2,RC
1797 CLOSE #2
1798 OPEN "R",#1,"PRESENT.BIN",128
1799 FIELD 1,128 AS IA$
1800 RETURN
1810 '
1820 ****
1830 'SUBROUTINE TO INTERROGATE TERMINAL
1840 A7%=INP (93): A8%=INP (92)
1850 A7%=A7% AND 2: IF A7%=0 THEN 1940          'DATA READY TEST
1860 A8%=A8% AND 127                         'MASK PARITY BIT
1870 A9%=A8% XOR 127: IF A9%=0 THEN 1930 ELSE 1940 'DELETE TO CP/M
1880 CLOSE #1
1890 OPEN "R",#2,"FILTERED.BIN",128
1900 FIELD 2,128 AS IB$
1910 PUT #2,RC1
1920 CLOSE
1930   SYSTEM
1940 RETURN
1950 ****
1960 *****
1970 LPRINT: LPRINT: LPRINT "LINE 342": GOTO 2030
1980 *****
1990 LPRINT: LPRINT: LPRINT "LINE 1265": GOTO 2030
2000 *****
2010 LPRINT: LPRINT: LPRINT "LINE 2025": GOTO 2030

```

```

2020      ****
2030      LPRINT "KRLP1=";KRLP1%, "KPLP1=";KPLP1%,
"OUTLP1=";OUTLP1%, "INLP1="; INLP1%
2040      LPRINT "XB=";XB%, "YB=";YB%, "XE=";XE%, "YE=";YE%
2050      LPRINT "DC=";DC, "RC=";RC, "S1=";S1%, "S2=";S2%, "S3=";S3%
2060      LPRINT "RDG1=";RDG1%, "RDR1=";RDR1%, "RDB1=";RDB1%
2070      LPRINT "RDG2=";RDG2%, "RDR2=";RDR2%, "RDB2=";RDB2%
2080      LPRINT "WRG1=";WRG1%, "WRR1=";WRR1%, "WRB1=";WRB1%
2090      LPRINT "WRG2=";WRG2%, "WRR2=";WRR2%, "WRB2=";WRB2%
2100      LPRINT "WS1=";WS1%, "WS2=";WS2%, "WS3=";WS3%
2110      RETURN
2120
'*****
2130      LPRINT : LPRINT : LPRINT "LINE 236"
2140      LPRINT "W1=";W1%, "W2=";W2%, "W3=";W3%
2150      LPRINT "W4=";W4%, "W5=";W5%, "W6=";W6%
2160      LPRINT "W7=";W7%, "W8=";W8%, "W9=";W9% : RETURN
2170
'*****
2180      'DEFINE RECORD NO.
2190      RC = ((OUTLP1%-1)*4)+1           ' 4-RECORDS/LINE, RECORD-
1 TO RECORD-4
2200      IF (X>1) AND (X<=2) THEN RC=RC+1  'CHECK FOR PIXEL
OVERFLOWING A RECORD
2210      IF (X>2) AND (X<=3) THEN RC=RC+2
2220      IF (X>3) THEN RC=RC+3
2230      RETURN
2240      END

```



BASIC PROGRAM LISTING

DIS.ASC



200 PRINT: PRINT "FILE: DIS.ASC, REV. 10/7/4 19:00"  
 220 'ADDED ANNOTATIONS AND DELETED UNNECESSARY MATERIAL  
 240 CLEAR  
 260 INPUT "MURPHY (M) OR CAMILLE (C)"; K2\$ 'SELECT SYSTEM CONFIGURATION  
 280 PRINT: PRINT "JOYSTICK BIAS VALUES" 'CALIBRATE JOYSTICKS  
 300 E%  
 320 FOR A% = 2 TO 8 STEP 2: B% = A% - 2: OUT 236, B%: C% = INP (237): GOSUB 440  
 340 NEXT A%  
 360 IF K2\$ = "C" THEN 400  
 380 A8% = INP (1): GOTO 420 'KEYBOARD EXIT OF  
 JOYSTICK CALIBRATE  
 400 A8% = INP (93)  
 420 A8% = A8% AND 2: IF A8% = 0 THEN 320 ELSE 660  
 440 \*\*\*\*\*  
 460 'SUBROUTINE TO ACQUIRE JOYSTICK VALUES  
 480 D% = A%/2: ON D% GOTO 500, 520, 540, 580  
 500 PRINT "SCALE="; C%, : BS5% = C%: GOTO 620  
 520 PRINT "X=" ; C%, : BX5% = C%: GOTO 620  
 540 PRINT "ANGLE="; C%, : BA5% = C%: GOTO 620  
 580 PRINT "Y=" ; C% : BY5% = C%  
 600 PRINT CHR\$(11);  
 620 RETURN  
 640 \*\*\*\*\*  
 660 ' PRINT: PR1% = 1: PR2% = 1: PR5% = 128: DB1% = 15  
 680 PR7\$ = "N" 'INPUT "PRINTOUT? 'Y' OR 'N'" ; PR7\$  
 700 PR9\$ = "Y" 'INPUT "UPDATE EACH FIELD, NOT ALTERNATE  
 FIELDS; 'Y' OR 'N"'; PR9\$  
 720 PR10\$ = "Y" 'INPUT "CLEAR REMAINDERS; 'Y' OR 'N"'; PR10\$  
 740 PR11\$ = "Y" 'INPUT "SLOPE USING CINT ROUNDOFF; 'Y' OR 'N"'; PR11\$  
 760 INPUT "SCALE FACTOR; MIN, MAX; .1 TO 1500; 10,600 IS  
 NOMINAL"; PR12, PR13  
 780 IF PR12 = 0 OR PR13 = 0 THEN 800 ELSE 820  
 800 PR12 = 10: PR13 = 600  
 820 IF PR9\$ = "Y" GOTO 860 'INTERPOLATE ONCE EACH FIELD OR  
 ONCE EACH FRAME  
 840 TS1 = 1/4: GOTO 880 'NUMBER OF TIME SLICES  
 860 TS1 = 1/8  
 880 INPUT "NEW WINDOW GEOMETRY: 'Y' OR 'N"'; PR32\$  
 900 PR16\$ = "F" 'INPUT "INTEGER (I) OR NON-INTEGER (F)  
 TRANSLATION"; PR16\$  
 920 IF PR32\$ = "Y" GOTO 960  
 940 PRINT: INPUT "OFFSET CENTER OF ROTATION: TX, TY"; TX, TY: GOTO 980  
 960 PRINT: INPUT "OFFSET CENTER OF IMAGE: X1, Y1"; X1, Y1  
 980 DB1% = 20 'PRINT: INPUT "DEADBAND SELECTION; 20 IS NOMINAL"; DB1%  
 1000 PR21\$ = "Y" 'PRINT: INPUT "FRACTIONAL INITIAL POINT AND  
 SLOPES, Y OR N"; PR21\$  
 1020 DR = .017453292#: RD = 57.29577951000028#  
 1040 X5V = 512: Y5V = 482 'VIEWPORT DIMENSIONS  
 1060 '\*\* ADAPT X5V FOR PIXEL RATE; (54 US) (9.15 MHZ) (16/18 CLOCK WIDTH  
 1080 X5I = 512: Y5I = 512 'IMAGE MEMORY DIMENSIONS IN PIXELS  
 1100 'A R1 = SQR(X5V^2 + Y5V^2) 'UNITS OF PIXELS  
 1120 'A AP1 = ATN(Y5V/X5V)  
 1140 'A A6 = ATN(X5V/Y5V): AP1 = 90 \* DR - A6  
 1160 KS1% = X5I \* 8: KS2% = Y5I \* 8 'IMAGE MEMORY DIMENSIONS IN  
 EIGHTH PIXELS

```

1180      Q2=(Y5V/2)-TY: Q3=(X5V/2)+TX
1200      IF PR32$="Y" GOTO 1260
1220      AP1=ATN(Q2/Q3): R1=2*SQR(Q2^2+Q3^2)
1240      KB1=R1*8*SIN(AP1)/2: KB2=R1*8*COS(AP1)/2: GOTO 1280
1260      KB1=Q2*8: KB2=Q3*8
1280      DS11=1: JSS%=128
1300      XC1=(X5I/2+X1)*8: YC1=(Y5I/2+Y1)*8: TX=TX*8: TY=TY*8
1320      F6%=1: F7%=0
1340      XSV=1.58           'VIEWPORT ASPECT RATIO
1360      IF PR32$="Y" GOTO 1420
1380      XS=256*XSV : YS=256: YSS=YS*(3.90625E-03):
XSVR=1/(XSV*256)
1400      KSAR=0: KCAR=1: XSSV=XS*XSV: GOTO 1460
1420      XS=256*XSV : YS=256: YSS=1
1440      KSAR=0: KCAR=1: XSSV=1
1460      FOR E%=1 TO 16: GOSUB 2220: NEXT E%      'INITIAL
CONDITION GENERATION
1480      R%=INP (236): S%=R% AND 1: IF S%=1 THEN 1480      'LOCKUP
ON VERT.SYNC=1
1500      R%=INP (236): S%=R% AND 16: IF S%=0 THEN 1480      'CHECK FIELD
1520      'ITERATIVE PROCESSING
1540      OUT 236,64
1560      'RESYNCHRONIZATION AND FIELD CONTROL PROCESSOR
1580      R%=INP (236): S%=R% AND 1
1600      IF S%=0 THEN 1580      'LOCKUP ON VERT.SYNC=0
1620      'INTERLACED SCAN CALCULATIONS
1640      'INPUT BYTE    128 064 032 016 008 004 002 001
1660      '                           F2     F1     LS     FS
1680      OUT 236,0          'COMMAND LOAD, RUN-BAR
1700      R%=INP (236): S%=R% AND 16: IF S%=0 THEN 1860 ELSE 1720
'CHECK FIELD
1720      'FIELD-2
1740      'OUTPUT POSITION PARAMETERS
1760      OUT 238, 249: OUT 237, CA2%: OUT 236,128: OUT 236,0 'Y-ROW MSH
1780      OUT 238, 250: OUT 237, CB2%: OUT 236,128: OUT 236,0 'Y-ROW LSH
1800      OUT 238, 246: OUT 237, CC2%: OUT 236,128: OUT 236,0 'X-ROW MSH
1820      OUT 238, 247: OUT 237, CD2%: OUT 236,128: OUT 236,0 'X-ROW LSH
1840      GOTO 1980
1860      'FIELD-1
1880      'OUTPUT POSITION PARAMETERS
1900      OUT 238, 249: OUT 237, CA1%: OUT 236,128: OUT 236,0 'Y-ROW MSH
1920      OUT 238, 250: OUT 237, CB1%: OUT 236,128: OUT 236,0 'Y-ROW LSH
1940      OUT 238, 246: OUT 237, CC1%: OUT 236,128: OUT 236,0 'X-ROW MSH
1960      OUT 238, 247: OUT 237, CD1%: OUT 236,128: OUT 236,0 'X-ROW LSH
1980      'OUTPUT SLOPE PARAMETERS
2000      OUT 238,242: OUT 237,XPM%: OUT 236,128: OUT 236,0
'X-PIXEL SLOPE MSH
2020      OUT 238,245: OUT 237,YPM%: OUT 236,128: OUT 236,0
'Y-PIXEL SLOPE MSH
2040      OUT 238,248: OUT 237,XRM%: OUT 236,128: OUT 236,0
'X-ROW SLOPE MSH
2060      OUT 238,251: OUT 237,YRM%: OUT 236,128: OUT 236,0
'Y-ROW SLOPE MSH
2080      OUT 238,243: OUT 237,XPL%: OUT 236,128: OUT 236,0
'X-PIXEL SLOPE LSH

```

```

2100      OUT 238,244: OUT 237,YPL%: OUT 236,128: OUT 236,0
'Y-PIXEL SLOPE LSH
2120      OUT 238,240: OUT 237,XRL%: OUT 236,128: OUT 236,0
'X-ROW SLOPE LSH
2140      OUT 238,241: OUT 237,YRL%: OUT 236,128: OUT 236,0
'Y-ROW SLOPE LSH
2160      OUT 236,80  'COMMAND RUN, LOAD-BAR ;PULSE-1 BRACKETING
COMPUTATION PERIOD
2180      GOSUB 2220          'PROCESSING FOLLOWS OUTPUT
2200      GOTO 1520          'LOOP BACK FOR NEXT FIELD
2220  *****SUBROUTINE FOR INTERFIELD PROCESSING
2240      F6%=F6%+1          'INCREMENT TIME SLICE COUNTER
2260      ON F6% GOTO 4000, 4640, 4800, 2280, 2820, 3360, 3440,
3540
2280  ****
2300      GOSUB 5220: GOSUB 5840  'INTERPOLATE
2320      'CALCULATE INITIAL POINT
2340      XIP1N%=XC1+TX-KF6-KF4    'EQUIVALENT TO XIP1N%=XC1-
R1*8*COS(A5%)/2
2360      IF XIP1N%<0 THEN 5180 ELSE 5220
2380      XIP1N%=XIP1N%+KS1%: GOTO 5300  'WRAP AROUND
2400      IF XIP1N%>KS1% THEN 5260 ELSE 5300
2420      XIP1N%=XIP1N%-KS1%
2440      YIP1N%=YC1+TY+KF7-KF5 'EQUIVALENT TO YPI1%=YC1-R1*8*SIN(A5%)/2
2460      IF YIP1N%<0 THEN 5380 ELSE 5420
2480      YIP1N%=YIP1N%+KS2%: GOTO 5500  'WRAP-AROUND
2500      IF YIP1N%>KS2% THEN 5460 ELSE 5500
2520      YIP1N%=YIP1N%-KS2%
2540      XIP1N%=XIP1N%-XRN%  'BUFFER MEMORY WITH ANTI-ALIASING
2560      YIP1N%=YIP1N%-YRN%
2580      XIP2N%=XIP1N%+XRN%*(.015625)  'XIP1-SF=8, XR%-SF=256;
1/2*(8/256)=1/64
2600      YIP2N%=YIP1N%+YRN%*(.015625)
2620      'JOYSTICK PROCESSOR; MUST OCCUR MORE THAN 100-
MICROSECONDS AFTER FRAME SYNC GOES LOW
2640      OUT 236,66: JSXV%=INP (237): OUT 236,68: JSA%=INP (237)
2660      OUT 236,70: JSYV%=INP (237): OUT 236,64: JSS%=INP (237): OUT 236,
2680      JSXB%=JSXV%-BX5%: JSYB%=JSYV%-BY5%: JSSB%=JSS%-BS5%:
JSAB%=JSA%-BA5%
2700      IF ABS(JSAB%)<DB1% THEN 2800          'CHECK IF IN DEADBAND
2720      IF JSAB%<0 THEN 2740 ELSE 2760          'BIAS OUT DEADBAND
2740      JSAB%=JSAB%+DB1%: GOTO 2780
2760      JSAB%=JSAB%-DB1%
2780      AR=AR+JSAB%*ABS (JSAB%)*(.00003)        'SQUARE LAW JOY
STICK SCALING
2800      RETURN
2820  ****
2840      IF PR9$="N" GOTO 2880
2860      GOSUB 5220: GOSUB 5840  'INTERPOLATION
2880      GOSUB 4980          'KEYBOARD INPUT ROUTINE
2900      IF PR10$="N" THEN 2960
2920      SDXIP1R%=0: SDYIP1R%=0: SDXIP2R%=0: SDYIP2R%=0
2940      SDXPR%=0: SDYPR%=0: SDXRR%=0: SDYRR%=0
2960      DXIP1%=(XIP1N%-XIP1P%): DYIP1%=(YIP1N%-YIP1P%)
2980      DXIP2%=(XIP2N%-XIP2P%): DYIP2%=(YIP2N%-YIP2P%)

```

```

3000 DYP%=(YPN%-YPP%): DXP%=(XPN%-XPP%)
3020 DYR%=(YRN%-YRP%): DXR%=(XRN%-XRP%)
3040 'REMAINDER PROCESSING WORKS FOR + AND - DELTA NUMBERS
3060 IF PR9$="Y" GOTO 3180
3080 DXIP1R%=DXIP1% AND 3: DYIP1R%=DYIP1% AND 3
3100 DXIP2R%=DXIP2% AND 3: DYIP2R%=DYIP2% AND 3
3120 DXPR%=DXP% AND 3: DYPR%=DYP% AND 3
3140 DXRR%=DXR% AND 3: DYRR%=DYR% AND 3
3160 GOTO 3260
3180 DXIP1R%=DXIP1% AND 7: DYIP1R%=DYIP1% AND 7
3200 DXIP2R%=DXIP2% AND 7: DYIP2R%=DYIP2% AND 7
3220 DXPR%=DXP% AND 7: DYPR%=DYP% AND 7
3240 DXRR%=DXR% AND 7: DYRR%=DYR% AND 7
3260 DXIP1%=DXIP1%*TS1: DYIP1%=DYIP1%*TS1
3280 DXIP2%=DXIP2%*TS1: DYIP2%=DYIP2%*TS1
3300 DXP%=DXP%*TS1: DYP%=DYP%*TS1
3320 DXR%=DXR%*TS1: DYR%=DYR%*TS1
3340 RETURN
3360 ****
3380 GOSUB 5220: GOSUB 5840 'INTERPOLATE
3400 KSAR=SIN(AR)
3420 RETURN
3440 ****
3460 IF PR9$="N" GOTO 3500
3480 GOSUB 5220: GOSUB 5840 'INTERPOLATE
3500 KCAR=COS(AR)
3520 RETURN
3540 ****
3560 F6%=0 'RESET TIME SLICE COUNTER
3580 GOSUB 5220: GOSUB 5840 'INTERPOLATE, FIRST OPERATION IN
PRIOR TIME SLICE
3600 JSYK%=JSYB%*KSAR: JSXK%=JSXB%*KSAR
3620 JSXB%=JSXB%*KCAR+JSYK%: JSYB%=JSYB%*KCAR-JSXK%
3640 'UPDATE POSITION; PRECEDES XIP, YIP PROCESSING
3660 IF ABS(JSXB%)<DB1% THEN 3980
3680 ' IF XC1>30000 AND JSXB%<0 THEN 7420 'LIMIT X-
TRANSLATION, SCROLLING
3700 ' IF XC1<-30000 AND JSXB%>0 THEN 7420
3720 IF XC1>30000 THEN 3740 ELSE 3760 'LIMIT X-
TRANSLATION, WRAP-AROUND
3740 XC1=XC1-32760: GOTO 3800
'(512*8)PIXELS*8 SUBPIXELS-8 SUBPIXELS
3760 IF XC1<-30000 THEN 3780 ELSE 3800
3780 XC1=XC1+32760
3800 IF JSXB%<0 THEN 3820 ELSE 3840 'BIAS OUT DEADBAND
3820 JSXB%=JSXB%+DB1%: GOTO 3860
3840 JSXB%=JSXB%-DB1%
3860 IF PR16$="F" THEN 3960
3880 DX1=DXR1+JSXB%*ABS(JSXB%)*(.075)
3900 DXI1=FIX(DX1)
3920 DXR1=DX1-DXI1
3940 XC1=XC1-DXI1: GOTO 3980
3960 XC1=XC1-JSXB%*ABS(JSXB%)*(.075)
3980 RETURN
4000 ****

```

```

4020    OUT 236,65      'OUTPUT START PULSE, INCREMENT TIME SLICE COUNTER
4040    IF PR9$="N" GOTO 4080
4060          GOSUB 5220: GOSUB 5840  'INTERPOLATE
4080    IF ABS(JSYB%)<DB1% THEN 4400
4100        IF YC1>30000 AND JSYB%>0 THEN 7740      'LIMIT Y-
TRANSLATION, SCROLLING
4120        IF YC1<-30000 AND JSYB%<0 THEN 7740
4140        IF YC1>30000 THEN 4160 ELSE 4180      'LIMIT X-
TRANSLATION, WRAP-AROUND
4160            YC1=YC1-32760: GOTO 4220
4180            IF YC1<-30000 THEN 4200 ELSE 4220
4200            YC1=YC1+32760
4220            IF JSYB%<0 THEN 4240 ELSE 4260      'BIAS OUT DEADBAND
4240            JSYB%=JSYB%+DB1%: GOTO 4280
4260            JSYB%=JSYB%-DB1%
4280            IF PR16$="F" THEN 4380
4300                DY1=DYR1+JSYB%*ABS(JSYB%)*(.075)
4320                DYI1=FIX(DY1)
4340                DYR1=DY1-DYI1
4360                YC1=YC1-DYI1: GOTO 4400
4380                YC1=YC1+JSYB%*ABS(JSYB%)*(.075)
4400    IF ABS(JSSB%)<DB1% THEN 4620
4420        IF JSSB%<0 THEN 4440 ELSE 4460
4440        JSSB%=JSSB%+DB1%: GOTO 4480      'BIAS OUT DEADBAND
4460        JSSB%=JSSB%-DB1%
4480        DS11=1+JSSB%*ABS(JSSB%)*(0.00001)      'OFFSET
BINARY TO SIGN BINARY, SCALE, BIAS ABOUT UNITY
4500    IF DS11<1 AND XS<PR12 OR DS11>1 AND XS>PR12 THEN 4620
4520    IF DS11>1 AND XS>PR13 OR DS11<1 AND XS<PR13 THEN 4620
4540    XS=XS*DS11: YS=YS*DS11
4560    IF PR32$="Y" GOTO 4600
4580    YSS=YS*(3.90625E-03): XSSV=XSVR: GOTO 4620
4600    KB1=KB1*DS11: KB2=KB2*DS11
4620    RETURN
4640    ****
4660    GOSUB 5220: GOSUB 5840  'INTERPOLATE, CALCULATIONS
PERFORMED IN PRIOR TIME-SLICE
4680    IF PR32$="Y" GOTO 4740
4700    KF6=KB2*KCAR*XSSV: KF7=KB1*KCAR*YSS
4720    KF4=KB1*KSAR*XSSV: KF5=KB2*KSAR*YSS: GOTO 4780
4740    KF6=KB2*KCAR: KF7=KB1*KCAR
4760    KF4=KB1*KSAR: KF5=KB2*KSAR
4780    RETURN
4800    ****
4820    IF PR9$="N" GOTO 4860
4840        GOSUB 5220: GOSUB 5840      'INTERPOLATE
4860    IF PR11$="N" THEN 4920
4880    YPN%=CINT(KSAR*XS): XRN%=CINT(2*KSAR*YS)
'CALCULATE SLOPES
4900    YRN%=-CINT(2*KCAR*YS): XPN%=CINT(KCAR*XS): GOTO 4960
4920    YPN%=KSAR*XS: XRN%=2*KSAR*YS      'CALCULATE SLOPES
4940    YRN%=-2*KCAR*YS: XPN%=KCAR*XS
4960    RETURN
4980    ****
5000    IF K2$="C" THEN 5040      'KEYBOARD COMMANDS

```

```

5020    A7% = INP (1): A8% = INP (0): GOTO 5060
5040    A7% = INP (93): A8% = INP (92)
5060    A7% = A7% AND 2: IF A7% = 0 THEN 5200      'DATA READY TEST
5080    A8% = A8% AND 127                          'MASK PARITY BIT
5100    A9% = A8% XOR 27: IF A9% = 0 THEN 240      'ESCAPE TO MENU
5120    A9% = A8% XOR 74: IF A9% = 0 THEN 5140 ELSE 5160  '"J" TO
RECALIBRATE JOYSTICKS
5140        BS5% = JSS%: BX5% = JSXV%: BY5% = JSYV%: BA5% = JSA%: GOTO
5200
5160    A9% = A8% XOR 127: IF A9% = 0 THEN 5180 ELSE 5200  'DELETE TO CP/M
5180        SYSTEM
5200    RETURN
5220    ****
5240    'INTERPOLATION SUBROUTINE-1
5260    'UPDATE INITIAL POSITIONS AND SLOPES
5280    'REMAINDER PROCESSING
5300    SDXIP1R% = SDXIP1R% + DXIP1R%: SDYIP1R% = SDYIP1R% + DYIP1R%
5320    SDXIP2R% = SDXIP2R% + DXIP2R%: SDYIP2R% = SDYIP2R% + DYIP2R%
5340    SDXPR% = SDXPR% + DXPR%: SDYPR% = SDYPR% + DYP%
5360    SDXRR% = SDXRR% + DXRR%: SDYRR% = SDYRR% + DYRR%
5380    'INTERPOLATION DELTA UPDATES
5400    XIP1P% = XIP1P% + DXIP1%: YIP1P% = YIP1P% + DYIP1%
5420    XIP2P% = XIP2P% + DXIP2%: YIP2P% = YIP2P% + DYIP2%
5440    XPP% = XPP% + DXP%: YPP% = YPP% + DYP%
5460    XRP% = XRP% + DXR%: YRP% = YRP% + DYP%
5480    IF SDXIP1R% < 8 THEN 5520
5500        SDXIP1R% = SDXIP1R% - 8: XIP1P% = XIP1P% + 1
5520    IF SDYIP1R% < 8 THEN 5560
5540        SDYIP1R% = SDYIP1R% - 8: YIP1P% = YIP1P% + 1
5560    IF SDXIP2R% < 8 THEN 5600
5580        SDXIP2R% = SDXIP2R% - 8: XIP2P% = XIP2P% + 1
5600    IF SDYIP2R% < 8 THEN 5640
5620        SDYIP2R% = SDYIP2R% - 8: YIP2P% = YIP2P% + 1
5640    IF SDXPR% < 8 THEN 5680
5660        SDXPR% = SDXPR% - 8: XPP% = XPP% + 1
5680    IF SDYPR% < 8 THEN 5720
5700        SDYPR% = SDYPR% - 8: YPP% = YPP% + 1
5720    IF SDXRR% < 8 THEN 5760
5740        SDXRR% = SDXRR% - 8: XRP% = XRP% + 1
5760    IF SDYRR% < 8 THEN 5800
5780        SDYRR% = SDYRR% - 8: YRP% = YRP% + 1
5800
5820    RETURN
5840    ****
5850    'INTERPOLATION SUBROUTINE-2
5860    'FORMAT INITIAL POINT OUTPUTS
5880    IF PR21$ = "N" THEN 6020
5900    CA1% = YIP1P% * (.015625): CB1% = YIP1P% AND 63:
CC1% = XIP1P% * (.015625): CD1% = XIP1P% AND 63
5920    CA2% = YIP2P% * (.015625): CB2% = YIP2P% AND 63:
CC2% = XIP2P% * (.015625): CD2% = XIP2P% AND 63
5940    'FORMAT SLOPE OUTPUTS
5960    XPM% = XPP% * (.015625): XPL% = XPP% AND 63:
YPM% = YPP% * (.015625): YPL% = YPP% AND 63
5980    XRM% = XRP% * (.015625): XRL% = XRP% AND 63:

```

YRM% = YRP% \* (.015625): YRL% = YRP% AND 63  
6000 GOTO 6140  
6020 'FORMAT IP AND SLOPE AND TRUNCATE TO PIXEL RESOLUTION  
6040 CA1% = YIP1P% \* (.015625): CB1% = YIP1P% AND 56:  
CC1% = XIP1P% \* (.015625): CD1% = XIP1P% AND 56  
6060 CA2% = YIP2P% \* (.015625): CB2% = YIP2P% AND 56:  
CC2% = XIP2P% \* (.015625): CD2% = XIP2P% AND 56  
6080 XPM% = XPP% \* (.015625) AND 60: XPL% = XPP% AND 0:  
YPM% = YPP% \* (.015625) AND 60: YPL% = YPP% AND 0  
6100 XRM% = XRP% \* (.015625) AND 60: XRL% = XRP% AND 0:  
YRM% = YRP% \* (.015625) AND 60: YRL% = YRP% AND 0  
6120  
6140 RETURN  
6160 \*\*\*\*\*  
6180 END

DISCLOSURE DOCUMENT

A disclosure document has been filed in the U. S. Patent and Trademark Office on or about October 18, 1984 No. 1,747, which is herein incorporated by reference. This disclosure document has copies of many of the documents and specification sheets referenced herein as follows.

1. National Semiconductor specification sheet for the MM5321 synchronization generator.
2. Signetics specification sheet for the 8T95, 96, 97, 98 hex buffers/inverters.
3. National Semiconductor specification sheet for the ADC 0800 A/D converter.
4. Texas Instruments specification sheet for the <sup>TMS</sup><sub>^</sub> TNS-4016 RAM.
5. TRW specification sheet for the TDC1016J-8, TDC1016J-9, TDC1016J-10 video D/A converters.
6. Mitsubishi Electric specification sheet for the M58725P, S;P-15, S-15, RAMs.
7. Computer Compatible Joystick Instruction sheet.
8. CompuPro CPU 8085/88 Technical Manual.
9. Viewpoint/3A Plus User's Manual.
10. CompuPro RAM 17 Technical Manual.
11. CompuPro RAM 16 Technical Manual.
12. CompuPro 8080 Multi-User Monitor program listing.
13. CompuPro System Support 1 User Manual.
14. International Instrumentation Incorporated Universal Disk Enclosures manual.
15. Siemens OEM Floppy Disk Drive FDD 100-8 manual.

16. CompuPro Disk 1 User Manual.